Assignment 4 - Sequences

Submission instructions:

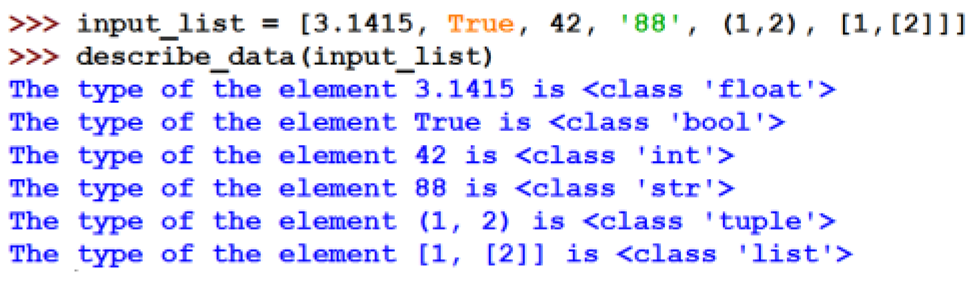
1. You cannot use other functions from other packages that are not given in the skeleton file
2. You must name your functions exactly as the questions stated.
3. The functions to be submitted should work for both tuples and lists. And you should NOT change the value of the input.
4. No mark will be given if the code cannot be compiled and run in coursemology.

Failure to follow each of the instruction will result in deduction of your marks.

Question 1: Describe Data [10 marks]

Write a function, describe_data, that takes a sequence L and prints each element within it along with its type.

Sample output:

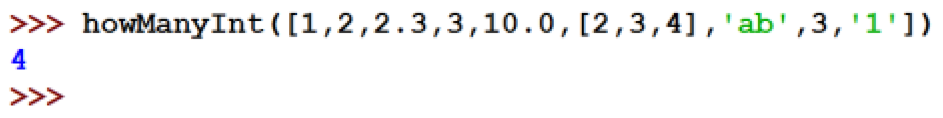


```
>>> input_list = [3.1415, True, 42, '88', (1,2), [1,[2]]]
>>> describe_data(input_list)
The type of the element 3.1415 is <class 'float'>
The type of the element True is <class 'bool'>
The type of the element 42 is <class 'int'>
The type of the element 88 is <class 'str'>
The type of the element (1, 2) is <class 'tuple'>
The type of the element [1, [2]] is <class 'list'>
```

Question 2: Counting integers [10 marks]

Write a function, howManyInt(), that takes in a sequence L. The function return how many integers are there in the list. Note that the number “10.0” is NOT an integer. Also, you do not need to go into a nested sequence.

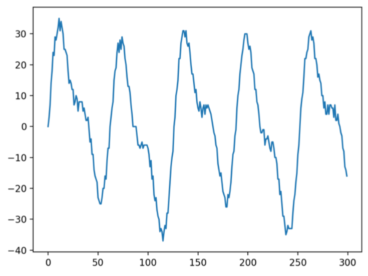
NOTE: No credit will be given if the inbuilt function “count” is used.



```
>>> howManyInt([1,2,2.3,3,10.0,[2,3,4],'ab',3,'1'])
4
>>>
```

Question 3: Filtering a wave [40 marks]

You are given a wave as a list. Your job is to smooth the wave out by a simple filtering technique. You are given a wave in a list named ‘`original_wave`’ as follows:



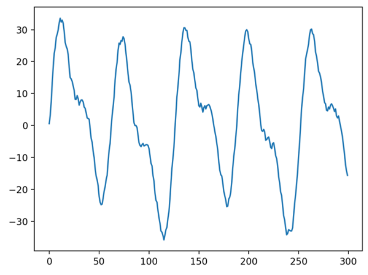
Write a function ‘`filter_wave(wave,1)`’ and this will produce a new wave which is a smoothed version of the input ‘wave’. Following the rules below

- In the new wave, for every position i, it is the weighted sum of three positions of the original wave. More preciously, the new wave value in position i will be equal to

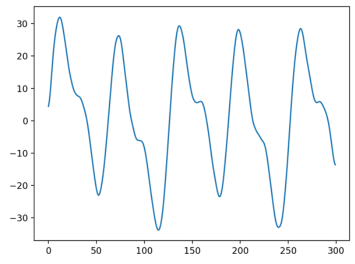
```
new_wave[i] = wave[i-1] * 0.2 + wave[i]*0.6 + wave[i+1]*0.2
```

- Let `len(wave)` be `L`. The above method will access `wave[-1]` and `wave[L]` in which do NOT exist in the original wave. So, you can deem the values of `wave[-1]` and `wave[L]` as `0`.
- You should **NOT** modify the original wave input
- And all the number in the new wave will be integers. You can simple use the function int() to convert any number into an integer.

Here is the expected shape of the wave after `filter_wave(original_wave_sample,1)`



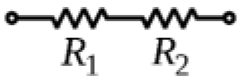
Finally, modify the function ‘`filter_wave(wave,n)`’ for **n** ≥ 0. And this will repeat the filtering **n** times to the wave accumulatively and produce an even smoother wave. Here is the expected wave for `filter_wave(original_wave_sample,10)`



Question 4: Finding Resistor Pairs [40 marks]

We are going to fly a lot of drones to scout out an area!

Everything is ready except that they need some resistors in their main circuits! For each drone, it needs two extra resistors that must be connected in series to produce the desired total resistance. Namely, total resistance R = R1 + R2.



For example, if you need a total resistance of 10 ohms, you can connect two resistors with resistances 2 ohms and 8 ohms, or another pair of 3 ohms and 7 ohms. You are given as many drones as possible, and they all need the same resistance R. In your hands, you are given a lot of resistors such that every resistor is unique and none of them has the same resistance with each other (Note that we will relax this uniqueness requirement in the later part). And of course, every resistor has a positive non-zero integer value of resistance. Your task is, given a tuple that contains the resistance values of all the resistors, find out all the pairs of resistors that can produce a total sum of R ohms. For example, if you have a tuple of resistors:

```
resistorList = (75,80,90,77,88,91,60,74,73,70,55,93,59)
```

You can call your function to find out what are the combinations for different total resistance. These two examples find all the resistors that can sum up to 150 ohms and 152 ohms.

```
>>> print(matchResistors(resistorList,150))
[(59, 91), (60, 90), (70, 80), (73, 77)]
>>> print(matchResistors(resistorList,152))
[(59, 93), (75, 77)]
```

Standard (20 marks)

Write the function 'matchResistors(R,n)' to return a list of tuples L. And each of the tuple contains a pair of resistor

in the list L that sum up to R like the examples above.

Again, here are some rules for your function in order to gain full marks:

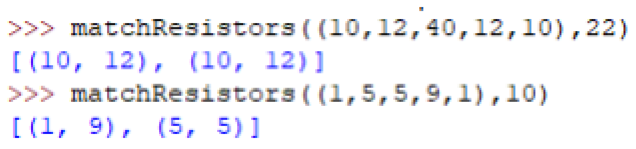
- You cannot modify the original input. And your function should work with inputs of both lists and tuples. You cannot hardcode the input
- You cannot import any packages or functions but you may use any of the built-in functions and methods. You may use sorted() function and .sort() method freely. However, you cannot use any set operations. (Believe us, that is not helpful at all.)
- Every item in your output must be unique (in this part). Namely, you cannot output the same pair twice, even if you swap the two values.
- Your output may be in a different order, and that is ok. Coursemology checker (i.e., test_resistors) is doing the job of checking any possible correct order for you.

The followings are two more tasks. We recommend you to make a backup for the above and just submit it in case you

cannot finish the following two parts.

Duplications (10 marks)

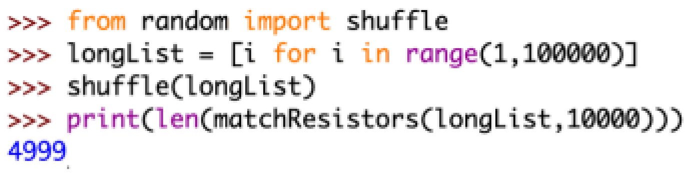
You will get this 5 marks if your function 'matchResistors(R,n)' can take in inputs with duplicate resistors. In other words, the input may contains two resistors with the same value. Note that each resistor can still only be used at most once. Hence, in the sample run below, once one of the 10 ohm resistor is used with a 12 ohm resistor, both are not available anymore but since there is still another 10 ohm and another 12 ohm, we can still get another pair. However, if your output has duplicates, they must have the correct number of copies.



```
>>> matchResistors((10,12,40,12,10),22)
[(10, 12), (10, 12)]
>>> matchResistors((1,5,5,9,1),10)
[(1, 9), (5, 5)]
```

Large Inputs (10 marks)

You get this 5 marks if your function 'matchResistors(R,n)' can handle large cases, e.g. the length of the input tuple > 1000000. (The function shuffle() is just used to create an example input list. Namely, it just randomly shuffle the list.)



```
>>> from random import shuffle
>>> longList = [i for i in range(1,100000)]
>>> shuffle(longList)
>>> print(len(matchResistors(longList,10000)))
4999
```