Exercise 1: Defying Gravity

Write a function that allows a user to introduce the distance d (in meters) travelled by an object falling, or the time t (in seconds) taken for an object to fall, but not both. The function should return the corresponding falling time t or distance d, respectively. The equations to be used are provided below:

$$d = \frac{1}{2}gt^2$$

$$t = \sqrt{\frac{2d}{g}}$$

where g is the gravitational acceleration and, for the sake of this exercise, it can be considered equal to $g = 9.81$ (m/s^2).

Complete function `freeFall(val,isD)`, taking as input a float, `val`, and a bool, `isD`. When `isD==True`, `val` represents the value of d; otherwise, `val` represents the value of t. The function should return the result of the appropriate equation given above, rounded to the second decimal digit.

Examples:

- `freeFall(1,true)` should return `0.45`.
- `freeFall(0.45,false)` should return `0.99`.

Exercise 2: Rock-Paper-Scissor

Rock-Paper-Scissors (RPS, in short) is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are *rock* (a closed fist), *paper* (a flat hand), and *scissors* (a fist with the index finger and middle finger extended, forming a V). A game has only two possible outcomes: a draw, or a win for one player and a loss for the other. A player who decides to play *rock* will beat another player who has chosen *scissors* ("rock crushes scissors"), but will lose to one who has played *paper* ("paper covers rock"); a play of *paper* will lose to a play of *scissors* ("scissors cuts paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie (Wikipedia contributors, 2021d).

Write a function that, given a player's strategy for a series of games of RPS, will return the winning strategy.

Function `RPS(s)` takes as argument a string `s`. The string can only contain the letters `R`, `P`, or `S`, representing *rock*, *paper*, and *scissor*, respectively. Each letter corresponds to the shape chosen by the other player on each game. The function should return a string of shapes that wins every single game.

Examples:

- `RPS('RPS')` should return `'PSR'`.
- `RPS('PRSPRR')` should return `'SPRSPP'`.

Exercise 3: List to String

In this exercise you are required to convert an input list into a string. The input list can only contain two types of items: a single letter (i.e., A-Z and a-z, case sensitive) or another list with the same properties. The output string should be constructed in such a way that:

- It starts and ends with square brackets (i.e., `[` and `]`).
- The items of the list that are letters are simply concatenated to the string.
- The content of a sublist should be placed in between square brackets.

Complete function `list2str(l)`, which takes as input a list satisfying the above properties and provides as output a string, as described above.

Examples:

- `list2str([‘a’,[‘b’,‘c’]])` should return `‘[a[bc]]’`.
- `list2str([‘a’,[‘b’,[‘c’]]])` should return `‘[a[b[c]]]’`.

Please, note that the output of the function is a string, rather than a list, as it is quoted.

Exercise 4: Text Preprocessing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them (Wikipedia contributors, 2021c).

Traditionally, before applying advanced NLP techniques, a document must be preprocessed.

Complete function `textPreprocessing(text)` which applies the following preprocessing pipeline to a string `text`:

1. Removal of all punctuation marks `.?!,:;-[]{}()’"`. E.g., the string `‘Hi! The cats are playing’` becomes `‘Hi the cats are playing’`).
2. Conversion of text to lower case (e.g., `‘hi the cats are playing’`).
3. Segmentation into a list of words (e.g., `[‘hi’,‘the’,‘cats’,‘are’,‘playing’]`).
4. Removal of stopwords. Stopwords are words that are so common that do not much information and, therefore, can be removed. For this exercise, make use of the following list of stopwords:

   i, a, about, am, an, are, as, at, be, by, for, from, how, in, is, it, of, on, or, that, the, this, to, was, what, when, where, who, will, with

   (e.g., the result of this step on the sample sentence would be `[‘hi’,‘cats’,‘playing’]`)
5. Stemming, which aims at reducing different forms of a word into a common base form. For the sake of this exercise, you are required to apply a very crude stemmer, based on the following rule: if a word ends in `-ed`, `-ing`, or `-s`, remove the ending (e.g., `[‘hi’,‘cat’,‘play’]`).

The function should return the result as a list of strings.
Examples:

- `textPreprocessing('I think, therefore I am.')` should return `['think','therefore']`.
- `textPreprocessing('When life gives you lemons, make lemonade.')` should return `['life','give','you','lemon','make','lemonade']`.

Exercise 5: Dictionary Dominance

Complete function `isGreaterThan(dict1,dict2)` that takes as input two dictionaries in Python, or two Objects in JavaScript, having only string keys and numerical values. The function returns `True` if and only if `dict1` is greater than or equal to `dict2` with respect to all the keys, and it is strictly greater than `dict2` in at least one key.
Examples:

- `dict1={'a':1,'b':2}` and `dict2={'a':1,'b':1}`. In this case, `dict1` is equal to `dict2` with respect to `a`, but it is greater with respect to `b`; therefore, the function should return `True`.
- `dict1={'a':1,'b':1}` and `dict2={'a':1,'b':1}`. In this case, `dict1` and `dict2` are equivalent; therefore, the function should return `False`.
- `dict1={'a':1,'b':0}` and `dict2={'a':0,'b':1}`. In this case, `dict1` is greater than `dict2` with respect to `a`, but it is lower with respect to `b`; therefore, the function should return `False`.

The two dictionaries/objects might not necessarily have the same keys. If a dictionary/object does not have a key that the other one has, the former is lower than the latter with respect to that key, regardless of the value that the latter might have.
Examples:

- `dict1={'a':1,'b':2,'c':-10}` and `dict2={'a':1,'b':1}`. `dict1` is greater than `dict2` with respect to all the keys; therefore, the function should return `True`.
- `dict1={'a':1,'b':1}` and `dict2={'c':0}`. In this case, `dict1` is not greater than `dict2`, as `dict1` does not have the key `c`; therefore the function should return `False`.

Exercise 6: Reading CSV Files

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas (Wikipedia contributors, 2021b).

Write function `CSVsum(filename)` that reads a CSV file with or without a header, and provides as output a list of the sums of each column.

Example: Suppose that the following is the content of file `test.csv`:

```
var1,var2,var3
1.0, 2.0, 3.0
4.0, 1.0,3.0
0.0,5.0, 0.0
```

which translates into the following table:

var1	var2	var3
1.0	2.0	3.0
4.0	1.0	3.0
0.0	5.0	0.0

Then, `CSVsum('test.csv')` should return `[5.0,8.0,6.0]`.

Exercise 7: String to List

This exercise is basically the opposite of Exercise 3.

You are now required to convert an input string of letters and square brackets (i.e., `[` and `]`) into a list of letters and lists. The square brackets identify where a list starts and ends, while each letter translates into an element of the corresponding list. Read the description of Exercise 3 and see the examples below for more information.

Complete function `str2list(l)`, which takes as input a string satisfying the above properties and provides as output the corresponding list.

Examples:

- `str2list('[abc]')` should return `['a','b','c']`.
- `str2list('[a[bc]]')` should return `['a',['b','c']]`.

Exercise 8: Spacemon Competition

Spacemons are spirits that come from other planets of our star system. When two spacemons meet, they feel the urge to fight until one or them is defeated. Warlocks conjure, tame, and train teams of spacemons, to make them compete in a spectacular tournament.

Note: the paragraph above is a work of fiction.

In this exercise, you are required to complete the function `spacemonSim(roster1,roster2)`, which simulates the result of a competition between two teams of spacemons, `roster1` and `roster2`; the function returns `True` if `roster1` wins, or `False` otherwise.

Disclaimer: no spacemon was harmed in the making of this exercise.

A spacemon is represented as a three-element tuple: `(planet, energy, power)` in Python, in JavaScript as a three-element array: `[planet, energy, power]` where `planet` represents the type of the spacemon, `energy` is its stamina, and `power` is its ability to reduce another spacemon's energy. A roster is simply a list of spacemons.

The `planet` of a spacemon is particularly important as certain types are stronger/weaker against others, as represented in Table 1.

att \ def	Mercury	Venus	Earth	Mars
Mercury	×1	×2	×1	×0.5
Venus	×0.5	×1	×2	×1
Earth	×1	×0.5	×1	×2
Mars	×2	×1	×0.5	×1

Table 1: Attack multipliers depending on type.

In the table, the rows correspond to the attacking spacemon, the columns correspond to the defending spacemon. The cells show the multiplier that must be applied to the attacker's `power` to determine how much energy the defender loses.

A competition is divided into one-on-one matches. Spacemons take part in the competition according to their position in the roster. Therefore, the first match is between the first spacemon of each roster. The spacemons take turns to attack, with the first roster always attacking first. When a spacemon attacks, the total damage inflicted on the opponent is: `damage = type_mult * power`, where `type_mult` is the multiplier specified in Table 1. The `damage` is then subtracted from the opponent's `energy`. If a spacemon's energy drops to 0 (or less), the spacemon is defeated and the match ends. Then, a new match starts between the winner of the previ-

ous match and the next spacemon in the opponent's roster. The winning spacemon does not recover any lost `energy` between matches; also, the first spacemon to attack is, again, the one from the first roster.

Example: Let us consider the following rosters.

- `roster1` is comprised of `('Earth',100,10)` and `('Earth',100,10)`.
- `roster2` is comprised of `('Mercury',80,10)` and `('Venus',80,10)`.

In the first match, the Earth spacemon defeats the Mercury spacemon; however, it loses 70 points of `energy`. In the second match, the former winner is defeated by the Venus spacemon, which receives 10 points of `damage`. Finally, the Venus spacemon wins the third match, losing 35 points of `energy`. The second roster wins the competition and, therefore, the function returns `False`.