Verkefni 2

Poker hands

In the game of poker, each player is dealt five cards. The cards are then given one of ten ranks, given in the table below, and described here.

Write a function `rank_hand` that takes a list of strings, describing cards, and outputs an integer, denoting the rank of the hand, as given by the table below. The cards are represented as a string of length two. The first character denotes the rank of the card. The cards 2 up to 9 are denoted by the characters `'2'` to `'9'`, and the ten, jack, queen, king and ace are denoted by `'T'`, `'J'`, `'Q'`, `'K'` and `'A'`, respectively. The second character of the string denotes the suit of the card. Hearts, spades, diamonds and clubs are denoted by `'H'`, `'S'`, `'D'` and `'C'`, respectively.

Table 1: List of poker hand rankings

Hand	Return value
Royal flush	9
Straight flush	8
Four of a kind	7
Full house	6
Flush	5
Straight	4
Three of a kind	3
Two pair	2
One pair	1
High card	0

Example

```
>>> rank_hand([ '3D', '2H', '3C', 'QS', '8D' ])
1
>>> rank_hand([ 'KD', 'KH', 'KC', 'TS', 'TD' ])
6
>>> rank_hand([ 'JD', 'KD', 'TD', 'QD', 'AD' ])
9
```

Countdown

Countdown is a british gameshow involving word and number puzzles. The Letters round, in a game of Countdown, proceeds as follows. The two contestants are given 9 letters and are given 30 seconds to create as long a word as possible, from the 9 letters. The word must be an english word, present in the Oxford English Dictionary, to be allowed.

Write a function `countdown` that takes two paramters, a string containing a filename and a string containing nine letters. The the file, given by the first parameter, contains a dictionary (or a list of words), with each word given

on a seperate line. The function returns a list of words, that are given in the dictionary, that can be created using the nine letters of the second parameters. The list should be given in alphabetical order and no word shorter than four letters should apper in the list.

All the characters in this problem are english letters given in lower case. The dictionary file can be obtained here.

Example

```
>>> countdown('path/to/file', 'pythonxyz')
['hont', 'hypo', 'hypt', 'onyx', 'phony', 'phyton', 'pnxt', 'pnyx', 'pont',
'ponty', 'pony', 'poxy', 'poynt', 'pynot', 'pyot', 'python', 'thon', 'tony',
'toph', 'typhon', 'typo', 'typy', 'tyyn', 'yont']
```

Insert operators

Write a function `insert_operators(seq, target)` that takes as parameters a list of integers, `seq`, and an integer, `target`. The function determines whether it's possible to create an expression, by inserting operators between the elements of `seq`, such that the value of the expression is `target`.

The available operators are

- the `+` operator, which behaves like the usual addition operator,
- the `-` operator, which behaves like the usual subtraction operator,
- the concatenation operator, which concatenates the number on the left side with the number on the right side.

If a solution is found, the function should return a string representing the solution. If no solution is found, then `None` should be returned. If multiple solutions exist, then any of them can be returned.

Example

```
>>> insert_operators([14,8,2,17,5,9],83)
"14+82-17-5+9=83"
>>> insert_operators([34,9,82,21,32],32850)
"34982-2132=32850"
>>> insert_operators([1,2,3],5)
None
```

Hangman

Hangman is a two player game that proceeds as follows. One player chooses a word and tells the other player how long it is by writing a dash (`-`) for each letter in the word. The second player's goal is then to discover which word the first player chose. He does this by guessing letters. If the letter is in the word, the first player writes the letter in its correct positions, by replacing the dashes. If the second player guesses incorrectly 10 times, the first player wins; otherwise, if the second player manages to discover the word, he wins.

As computer scientist we don't want to waste brainpower thinking of words, so what we do is guess the first few letters, and then write a program that matches the partially guessed word to all the words in the english dictionary.

Write a function `hangman(dict_file, state, guessed)` that takes three strings as parameters. The first string gives the path to a dictionary file, all_words.txt, that contains a list of all the words that can appear in the game, one word per line. The second string contains the state of the game. The state is represented by a sequence of dashes, where the dashes have been replaced for correctly guessed letters. So, for example, if the second player has correctly guessed the letters i and o, the state could be `'-------io-'`. The third string is a list containing all

the letters player two has guessed. The function returns a list of all the possible words the first player might have chosen.

Note that the word chosen by the first player is a valid word from the given dictionary that contains only lower case english letters.

Example

```
>>> hangman('path/to/file', 's-a--o--s', 'aeiosu')
['scaffolds', 'shaddocks', 'shamrocks', 'spanworms', 'staghorns', 'standoffs', 'starworts']
```

Valid ID-numbers

Write a function `valid` that takes a string of 10 digits. The function returns True if the ID-numbers is a valid ID-number of a person born between 1900 and 2099; False otherwise. To determine whether the ID-number is valid or not, you have to use the checksum (english version) of the ID-number. Furthermore, the first six digits of the ID-number must represent a valid date.

Example

```
>>> valid('0212862149')
True
>>> valid('1803442379')
False
```

Count names

Statistics Iceland (Hagstofa Íslands) provides data on how many Icelandic citizens have a given first and middle name (see https://hagstofa.is/talnaefni/ibuar/faeddir-og-danir/nofn/).

The complete data is provided as a JSON file, nofn.json. The data are represented with a list of objects, one object for each name. Each object contains three keys, `Nafn`, `Fjoldi1` and `Fjoldi2`. The key `Nafn` provides the name that object represents, `Fjoldi1` gives the number of citizens that have the specified name as a first name, and `Fjoldi2` the number that have that name as a middle name.

```
[
  ...

  {
    "Nafn": "Linddís",
    "Fjoldi1": 1,
    "Fjoldi2": 1
  },
  {
    "Nafn": "Damjan",
    "Fjoldi1": 5,
    "Fjoldi2": 0
  },
  {
    "Nafn": "Bjarki",
    "Fjoldi1": 915,
    "Fjoldi2": 475
  },
```

```
  {
    "Nafn": "Þórhildur",
    "Fjoldi1": 374,
    "Fjoldi2": 38
  },
  ...
]
```

Write a function function `count_names` that takes one string, `start` as a parameter. The function returns a tuple (a, b) where a and b are the numbers of citizens that have a first and middle name that starts with the string `start`, respectively.

Note

To avoid spamming Statistics Iceland with request, you can use download the JSON file from http://mooshak.ru.is/~python/names.json.

Example

```
>>> count_names('Bja')
(3267, 1494)

>>> count_names('Wat')
(8, 2)

>>> count_names('Snati')
(0, 0)
```

Days of movie stars

In this problem you are given two CSV files, derived from data obtained from The Internet Movie Database (IMDb).

The file cast.csv contains information on cast of movies. The columns are the following.

- **year**: The production year of the movie
- **name**: The name of the actor/actress
- **type**: If the current row contains information about an actor this field contains the string `actor`; otherwise `actress`
- **character**: The name of the character portrayed by the actor/actress
- **n**: The order in which the actor/actress appeared in the cast list of the movie

The file dates.csv contains information on release dates of movies for various countries. The columns are the following.

- **year**: The production year of the movie
- **country**: The country of release
- **date**: The release date for the specified country

Your task is to write a function `release_days(cast, dates, actors)` that takes three parameters, `cast` and `dates` are paths to the cast.csv and dates.csv files, respectively. The parameter `actors` contains a list of strings with names of actors/actresses.

The function should categorize movies that starred one of the actors/actresses, given in `actors`, by the day of the week the movie was released in the USA.

The function should return a dictionary. The keys of the dictionary are integers in the range from 1 to 7 (inclusive), where 1 denotes Monday, 2 denotes Tuesday, etc. The values of the dictionary are sets that contain the names of the movies that were released on the weekday corresponding to that set's key.

Example

```
>>> release_days('/path/to/cast.csv', /path/to/dates.csv', ['Meg Ryan', 'Tom Hanks'])
{2: {'Kate & Leopold'},
 3: {'A League of Their Own',
     'Catch Me If You Can',
     'Forrest Gump',
     'Innerspace',
     'Nothing in Common',
     'The Money Pit',
     'The Polar Express',
     'Toy Story',
     'Toy Story 2'},
 5: {'Addicted to Love',
     'Against the Ropes',
     'Amityville 3-D',
     'Anastasia',
     'Angels & Demons',
     'Apollo 13',
     'Armed and Dangerous',
     'Bachelor Party',
     'Big',
     'Bridge of Spies',
     'Captain Phillips',
     'Cars',
     'Cast Away',
     "Charlie Wilson's War",
     'City of Angels',
     'Cloud Atlas',
     'Courage Under Fire',
     'D.O.A.',
     'Dragnet',
     'Extremely Loud & Incredibly Close',
     'Flesh and Bone',
     'French Kiss',
     'Hanging Up',
     "He Knows You're Alone",
     'Hurlyburly',
     'In the Cut',
     'In the Land of Women',
     "Io sono l'amore",
     'Joe Versus the Volcano',
     'Larry Crowne',
     'Philadelphia',
     'Prelude to a Kiss',
     'Proof of Life',
     'Punchline',
     'Radio Flyer',
     'Restoration',
     'Road to Perdition',
     'Saving Mr. Banks',
```

```
     'Saving Private Ryan',
     'Serious Moonlight',
     'Sleepless in Seattle',
     'Splash',
     'Sully',
     'That Thing You Do!',
     "The 'Burbs",
     'The Bonfire of the Vanities',
     'The Da Vinci Code',
     'The Doors',
     'The Great Buck Howard',
     'The Green Mile',
     'The Ladykillers',
     'The Man with One Red Shoe',
     'The Presidio',
     'The Queen',
     'The Simpsons Movie',
     'The Terminal',
     'Top Gun',
     'Toy Story 3',
     'Turner & Hooch',
     'Volunteers',
     'When Harry Met Sally...',
     'When a Man Loves a Woman',
     "You've Got Mail"},
 7: {'I.Q.'}}
```

Process submissions

As programmers, we love to work with data. Getting data from different sources can vary in difficulty. Sometimes the source doesn't provide a neat way to get the data, or any kind of API. Then us programmers may need to get down and dirty and recover the data through other methods, for example by going through log files or other files created by the source.

Recently I was faced with such a problem. In one of the courses I teach, we use an online program evaluator to check the correctness of student solutions. I wanted to create an online scoreboard that displayed the students and their achievements. But as a first step to building the scoreboard, I had to go through a folder containing all the submissions to the program evaluator. The submissions folder looks as follows.

```
submissions

      submission_94533381
          data.tcl
          solution.py
      submission_11939229
          data.tcl
          hello.cpp
      submission_2348249
          data.tcl
          sol.cpp
      submission_29002
           data.tcl
           meow.c
```

We are interested in the `data.tcl` file in each submission directory. Each of them are of the following format.

```
set         Date 1373647394
```

```
set      Problem 2_remove_empty_strings
set         Team ghostbusters12
set     Classify Accepted
set         Mark 0
set         Size 208
set Observations {}
set    Execution -1
set        State final
set     Language Python
set      Program 2.py
set       Report 2.html
set      Elapsed 0
set          CPU 0
set       Memory 0
set     Feedback {}
```

Write a function `parse_submissions(directory)` that takes as a parameter a single string, the location of the submissions folder (i.e. the folder containing all the submissions). The function should return a list of tuples, each of them corresponding to a submission in the submissions directory. Each tuple contains two strings, the username of the student that owns the submission (which is specified in the Team field in the data.tcl file) and the name of the problem the submission belongs to (specified in the Problem field). The tuples should be sorted in ascending order by submission date (specified in the Date field). And finally, all submissions that do not have `Accepted` in the Classify field should be filtered out.

Example

The data for the example test case below can be found here.

```
>>> parse_submissions("submissions")
[('kKIPggAq', '2_remove_empty_strings'),
('xaYgfEco', '1_multiples_of_3_5'),
('sadGKWpL', '7_palindromes')]
```

Just a Minute

Just a Minute is a radio panel game that has been running since 1967. On each show, three (or sometimse more) panellists appear, along with a host. A list of the host and panellists of each show is given in the file jam.txt (the source of the list is here). The format of the file is as follows. First the date of each show is given, then the host and a list of panellists (and guests). Finally the topic of the show is given, if it is known.

Write a function `jam` that takes a string, containing the contents of jam.txt (or a subset of its contents) and returns a dictionary. The keys of the dictionary are the names of the panellists and hosts of the shows, and the value, for each key, is the number of times that person has appeared on the show.

Note that the test cases will all be from jam.txt. The final test case is the whole file.

Example

```
>>> jam("""1/1/1 22 December 1967, Nicholas Parsons with Derek Nimmo, Clement Freud, Wilma Ewart and Beryl
... 2/1/2 29 December 1967, Nicholas Parsons with Derek Nimmo, Clement Freud, Sheila Hancock and Carol Bin
... 3/1/3 5 January 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Betty Marsden and Elisabeth Be
... 4/1/4 12 January 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Isobel Barnett and Bettine Le
... 5/1/5 20 January 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Andree Melly and Prunella Sca
... 6/1/6 27 January 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Marjorie Proops and Millie Sm
... 7/1/7 2 February 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Aimi Macdonald and Una Stubbs
```

```
... 8/1/8 9 February 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Lucy Bartlett and Anona Winn,
... 9/1/9 17 February 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Andree Melly and Charmian In
... 10/1/10 23 February 1968, Nicholas Parsons with Derek Nimmo, Clement Freud, Barbara Blake and Renee Ho
{'Aimi Macdonald': 1,
 'Andree Melly': 2,
 'Anona Winn': 1,
 'Barbara Blake': 1,
 'Beryl Reid': 1,
 'Bettine Le Beau': 1,
 'Betty Marsden': 1,
 'Carol Binstead': 1,
 'Charmian Innes': 1,
 'Clement Freud': 10,
 'Derek Nimmo': 10,
 'Elisabeth Beresford': 1,
 'Isobel Barnett': 1,
 'Lucy Bartlett': 1,
 'Marjorie Proops': 1,
 'Millie Small': 1,
 'Nicholas Parsons': 10,
 'Prunella Scales': 1,
 'Renee Houston': 1,
 'Sheila Hancock': 1,
 'Una Stubbs': 1,
 'Wilma Ewart': 1}
```

CSS Properties

CSS is a language used to style HTML documents. An example of some CSS is as follows.

```
#LasVegas .billboard { text-decoration: blink; }

.ninja, #Snowden { visibility: hidden; }


.oliveoil
{
  z-index: 1;
}
.water
{
  z-index: 0;
}

#poop {
  float  : none  ;
  color  : brown ;

  width  : 15cm  ;
  height : 120cm ;
}

.God { position: absolute; display: none; }
#blackhole { padding: -9999em; }
```

```
.word {  font-family:    "Comic Sans", "Times New Roman", sans-serif  ;  }
```

CSS has two kinds of elements; selectors and properties. Selectors are expressions that select HTML elements from HTML document, and properties are key-value pairs that configure the style of the selected elements. In this problem we are only concerned with the properties. The following list shows all properties in the example CSS above.

```
text-decoration = blink
visibility = hidden
z-index = 1
z-index = 0
float = none
color = brown
width = 15cm
height = 120cm
position = absolute
display = none
padding = -9999em
font-family = "Comic Sans", "Times New Roman", sans-serif
```

Write a function `css_properties(css)` that takes as a parameter a single string containing CSS code, and returns all the properties found in it. The properties should be returned as a list of tuples, where each tuple contains the key and the value of the property. Note that the tuples should be in the same order as their respective properties are found in the code.

Example

```
>>> css_properties("""
#LasVegas .billboard { text-decoration: blink; }

.ninja, #Snowden { visibility: hidden; }


.oliveoil
{
  z-index: 1;
}
.water
{
  z-index: 0;
}

#poop {
  float  : none  ;
  color  : brown ;

  width  : 15cm  ;
  height : 120cm ;
}

.God { position: absolute; display: none; }
#blackhole { padding: -9999em; }

.word {  font-family:    "Comic Sans", "Times New Roman", sans-serif  ;  }
""")
[('text-decoration', 'blink'),
('visibility', 'hidden'),
```

```
('z-index', '1'),
('z-index', '0'),
('float', 'none'),
('color', 'brown'),
('width', '15cm'),
('height', '120cm'),
('position', 'absolute'),
('display', 'none'),
('padding', '-9999em'),
('font-family', '"Comic Sans", "Times New Roman", sans-serif')]
```

Scaling recipes

One common problem when working with recipes is scaling. If we have a recipe that is too big to fit our needs, we can, e.g., halve it (or scale it by a factor of 1/2). And vice versa, if we have a recipe too small to fit our needs we can, e.g., double it (or scale it by a factor of 2).

Scaling a recipe involves multiplying the amount of each ingredient by the given *scalar*. For example, if we were to scale the following recipe by 1/2 (or halve it):

- 4 skinless, boneless chicken thighs
- 1/2 cup soy sauce
- 1/2 cup ketchup
- 1/3 cup honey
- 3 cloves garlic, minced
- 1 teaspoon dried basil

We would obtain the following recipe:

- 2 skinless, boneless chicken thighs
- 1/4 cup soy sauce
- 1/4 cup ketchup
- 1/6 cup honey
- 1 1/2 cloves garlic, minced
- 1/2 teaspoon dried basil

Your task is to write a function `scale(recipe, scalar)` which takes two parameters, a string containing a recipe (or at least an ingredients list for a recipe) and a string containing a scalar, in that order. The function returns a string containing the given recipe after it has been scaled by the given scalar.

All numbers contained in `recipe` will be a part of the ingredients list. The string `scalar` will contain either an integer or a simple fraction denoting the scalar.

The numbers that appear in the recipe can be either integers, simple fractions (improper or proper) or mixed numbers (i.e. an integer followed by a simple fraction). The integer and the fraction in mixed numbers are separated by a single space.

The string returned by the function, should be identical to `recipe` in every way except the numbers in the returned string have been scaled by a factor of `scale`. The numbers in the returned string should be mixed numbers, where the fraction part has been reduced to lowest terms. If either the integer part of the fraction part is 0, they should **not** be part of the number.

Example

```
>>> scale('''Ingredients

    4 skinless, boneless chicken thighs
    1/2 cup soy sauce
```

```
    1/2 cup ketchup
    1/3 cup honey
    3 cloves garlic, minced
    1 teaspoon dried basil''', '1/2')
'''Ingredients

    2 skinless, boneless chicken thighs
    1/4 cup soy sauce
    1/4 cup ketchup
    1/6 cup honey
    1 1/2 cloves garlic, minced
    1/2 teaspoon dried basil'''
```