File Problem Set (Python)

Complete as much as you want, when you want. No rules on what you can lookup/google, though I do recommend spending more time googling how to do a specific operation you want then the problem itself, particularly for easier problems.

Problems here are specified as function headers which take input and with an expected output, though some problems may involve multiple functions if there's class design or similar involved.

Generally problems are intended to loosely get harder for later numbers, though obviously it depends on your comfort level with various concepts. I generally make the last problem or two quite a bit harder for fun though...

For this particular problem set, I include a few files as examples, though writing more is often a good idea. Note that there are assumptions made about the file structure in addition to the file extension.

References

A few things I assume, namely file formats that you can open. Note that file formats are *just conventions*, you can put anything inside any file you want (as always). If you use a given extension, however, you are assumed to follow the format standard to the letter (this isn't just me saying this, super standard expectation).

- `.fakeletters`: a completely made up format I invented to illustrate that extensions are arbitrary. The file format is a series of space-separated lowercase strings (that is, one or more lowercase letters). No other symbols may appear in this file format.
- `.csv`: CSV Files are extremely standard and useful to know about. Very simple file format
- `.json`: JSON Files extremely standard file used for storing data (though it should not be used as a database).

I decided not to write problems with JSON for now, but if you find you want to use them, I recommend using Python's tools for parsing

Problems

Count Words

```
def count_words(filename : str) -> int:
    """
    Returns the number of words in a given `.fakeletters` file

    A word consists of any number of letters

    Requires that `filename` ends with `.fakeletters`

```

```
    Examples:
      count_words("test1.fakeletters") -> 3
      count_words("test2.fakeletters") -> 2
      count_words("test3.fakeletters") -> 0
      count_words("test4.fakeletters") -> 69 (nice!)
    """
    pass
```

Contains

```
def contains(filename : str, word : str) -> bool:
    """
    Returns whether the given word appears in a `.fakeletters` file

    Requires that `filename` ends with `.fakeletters`

    Examples:
      contains("test1.fakeletters", "b") -> True
      contains("test1.fakeletters", "d") -> False
      contains("test2.fakeletters", "wor") -> False
      contains("test2.fakeletters", "hello") -> True
      contains("test3.fakeletters", "") -> False
      contains("test4.fakeletters") -> 69 (nice!)
    """
    pass
```

Write Fake Letters

```
def write_fake(filename : str, words : list[str]):
    """
    Writes the given list of strings to the `.fakeletters` file

    Each word must consist of only letters, and will be written
      to be all lowercase and space-separated

    (Note, you can use the previous two functions to help test this one!)

    Requires that `filename` ends with `.fakeletters`

    Example:
      write_fake("output.fakeletters", ["test", "out"])
      contains("output.fakeletters", "test") -> True
      contains("output.fakeletters", "other") -> False
      count_words("output.fakeletters") -> 2
    """
    pass
```

CSV

For all the following problems, please parse the csv file yourself. Note that there are tools to help you with this, but I'm asking you to not use them for the sake of learning + seeing that you don't need a magical library to do this.

Row Sum

```
def row_sum(filename : str) -> list[int]:
    """
    Returns the sum of each row in the given `.csv` file

    Requires that filename ends with `.csv`
    Also requires that the given CSV file consist of only integers

    Examples:
      row_sum("data1.csv") -> []
      row_sum("data2.csv") -> [4]
      row_sum("data3.csv") -> [2, -4, 18]
      row_sum("data4.csv") -> [6, 9, 12, 15]
    """
    pass
```

Column Mean

```
def column_mean(filename : str) -> list[float]:
    """
    Given a CSV filename, calculate the mean of each column in the CSV

    Requires that filename end with `.csv` and the file contains only integers

    Any empty row element is assumed to be 0

    Examples:
      column_mean("data1.csv") -> []
      column_mean("data2.csv") -> [3, -1, .5, 2]
      column_mean("data3.csv") -> [3, 1, 0]
      column_mean("data4.csv") -> [2.5, 3.5, 4.5]
    """
    pass
```

Transpose

```
def transpose(filename : str):
    """
    Given a CSV filename `example.csv`
```

```
    If that file is in the form of a matrix (each row has the same length)
      Writes the transpose of that file to `example_transpose.csv`
    Otherwise, this function does nothing

    A matrix transpose "flips" the matrix over the diagonal:
      https://en.wikipedia.org/wiki/Transpose

    Examples:
      transpose("data1.csv") produces an empty file `data1_transpose.csv`
      transpose("data2.csv") does nothing
      transpose("data3.csv") produces `data3_transpose.csv` with
        1,3,5
        -1,-2,6
        -2,-5,7
      transpose("data4.csv") produces `data4_transpose.csv` with
        1,2,3,4
        2,3,4,5
        3,4,5,6
    """
    pass
```

CSV Tables

For the remaining problems, we will be using CSV files as tables to show their flexibility as a representation. Note that this is not necessarily the best way to do things!

A table consists of the columns names on the first row, followed by the raw data. See the examples provided for reference.

Contains Name

```
def contains_name(filename : str, name : str) -> bool:
    """
    Given a filename ending with `csv`

    If that filecontains a `name` column, returns if `name` is in that column

    Examples:
      contains_name("data1.csv", "goblin") -> True
      contains_name("data1.csv", "Eve") -> False // uppercase
      contains_name("data2.csv", "bob") -> False // no name column
      contains_name("data3.csv", "Taylor Swift") -> True
      contains_name("data3.csv", "Sabaton") -> False
    """
    pass
```

Get Name Row

```
def get_name_total(filename : str, name : str) -> int:
    """
    Given a filename ending with `csv`

    Returns the total associated with `name`
    If the table has no `total` column or `name` is not found, returns -1

    If the file contains `first name` and `last name` rather than `name`
    Then this function should treat those as concatenated by a space

    Examples:
      contains_name("data1.csv", "goblin") -> -1 // no total
      contains_name("data2.csv", "bob smith") -> 1234
      contains_name("data2.csv", "alice jones") -> 6712
      contains_name("data3.csv", "Calvin Harris") -> 64953382
      contains_name("data3.csv", "The Weeknd") -> 108390904
      contains_name("data3.csv", "Test") -> -1
    """
    pass
```