DATA203 Foundational Python (Prof. Maull) / Fall 2025 / HW2

Points Possible	Due Date	Time Commitment (estimated)
25	Thursday, October 23 @ midnight	*up to* 15 hours

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by University or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

OBJECTIVES

- Practice writing functions.
- More practice writing functions and loading data from text files.
- Build a dictionary and understand and use files for data.

WHAT TO TURN IN

You will enjoy the highest benefits of the starter notebook if you clone the HW Github repository from your Jupyter Hub terminal with the command:

```
git clone https://github.com/kmhuads/f25_data203.git
```

This will ensure you have the most updated files and starter notebook.

Once you have cloned the repository, you can edit the starter notebook with your solution code.

When you are done with your work, it will be best to zip your `hw2` folder and all sub-folders with the terminal command (one level outside your notebook folder):

```
zip -r data203_hw2_maull.zip ./hw2`
```

This will produce the file with all necessary supporting files (notebooks, data output, etc.) then download it from the Jupyter Hub to your local machine, then upload the `.zip` to Teams.

If are confused on how to do this, please ask, or visit one of the many tutorials on the basics of using zip in Linux.

If you choose not to use the provided notebook, you will still need to turn in a `.ipynb` Jupyter Notebook and corresponding files according to the instructions in this homework.

ASSIGNMENT TASKS

(20%) Practice writing functions.

We can all use a little more practice writing functions in Python, so here we go.

In this part, we will input a list of numbers and determine, first, if there are any primes and second which of them (the primes) is largest.

As a quick reminder, prime numbers are interesting for a wide array of reasons (some practical, some purely mathematical), but a *prime number* is a number which is only divisible but 1 and itself. Common small primes we know, love and interact with everyday are 3, 5, 7, 11, 13, 17 and so on. It should be noted *finding* algorithms to search for large primes is hard work, and *not* something we will do with this assignment.

For example, do you know which number in the list below is prime? If there are more than one prime, which is the largest of them?

```
[834, 937, 1065, 1124, 1137, 1256, 1380, 1422, 1155, 1287, 1365]
```

We will dig in and practice how to do this in Python.

§ Task: 1.0 Write a function to the find the largest prime from a list of arbitrary size.

This should be very straightforward, but remember:

1. you will need to first check if a number is a candidate to be a prime number. Use the `is_prime()` function provided and this can be easily done. Also remember the following fact: *there are no even prime numbers*, or alternatively, if a number is divisible by 2, it *cannot* be prime,
2. *keep track* of the largest prime that you have seen, thus a loop will be your best tool along with a variable, say `max_prime` to store the largest seen so far,
3. make sure that if you see a new prime check it against your `max_prime`,
4. don't overthink it, you only need one loop and some conditional `if-else` logic to check the current number in the loop,
5. your function should be named `find_largest_prime`.

Here are some other bits of information you will find useful:

function **NAME**: `find_largest_prime`

function **INPUT**

- a Python list.

function **OUTPUT**

- `None` if there are no primes in the list,
- the largest prime number in the list.

Code Example:

```
input_list = [1, 2, 3, 4, 5]
largest_prime(input_list)
```

Output:

```
5
```

You will need to use the `is_prime()` function as provided in the starter notebook:

```
def is_prime(n):
    import math
    if n <= 1:
        return False
    else:
        is_prime = True
        for i in range(2, int(math.sqrt(n)) + 1):
            if n % i == 0:
                is_prime = False
                break
        return is_prime
```

Consult the notebook to make sure you are able to test your function against what is there. You will have to look at the outputs in the comments to check your work.

(50%) More practice writing functions and loading data from text files.

We learned in lecture that functions are necessary tools in Python and really all of computation.

In our Github repository you will notice there are two text files in `hw1/data/`:

- `data/test_data.csv`
- `data/story.txt`

You are encouraged to look at both files and see what is in them as they are different. Text files form the basis of many data file formats that you will interact with in your future as a data scientist.

In the notebook are some scaffolding code. You will need to study it and use it in the solutions being asked.

§ Task: 2.0 Practice explaining code.

Use the code in the first cell of the provided notebook and answer the question below:

1. Explain in your own words what function `mystery_function` in the cell does.

   Your explanation should include the description of the inputs, and you will need to review the Python file I/O libraries and specifically `readlines()` for details of the function.

§ Task: 2.1 Write a function that uses `mystery_function` and returns only the words that are 5 characters long.

In this task you will use `mystery_function` directly in a function of your own. Your function will take a single parameter `s` and return a list.

Give your function the name `find_len5_words()`.

Use this guidance to write your function:

function **NAME**: `find_len5_words`

function **INPUT**:

- `s` → an arbitrary Python string (paragraphs, etc).

function **OUTPUT**:

- an empty list `[]` if there are no words 5 characters long in `s`,
- otherwise the non-empty list of words in `s` which are 5 characters long.

Code Example:

```
input_string = mystery_function()
find_len5_words(input_string)
```

Output:

```
['five1', 'five2', 'five3'] # assuming these are the only words of length 5 in s
```

§ Task: 2.2 Rewrite and rename your `find_len5_words()` function so that it takes one additional parameter `w_size` (word size) which is a non-zero integer and returns all words in `s` of `w_size` length.

function **NAME**: `find_words`

function **INPUT**:

- `s` → an arbitrary Python string `s` (paragraphs, etc),
- `w_size` → a non-zero word size (`w_size`).

function **OUTPUT**:

- an empty list `[]` if there are no words `w_size` characters long in `s`,
- otherwise the non-empty list of words in `s` which are `w_size` characters long.

Code Example:

```
input_string = mystery_function()
find_words(input_string, 7) # return words of length 7
```

Output:

```
['word1ln', 'word2ln', 'word3ln'] # assuming these are the only words of length 7
```

Don't overthink this – your solution will be a very simple edit of your original `find_len5_words` code.

(30%) Build a dictionary and understand and use files for data.

We talked about file I/O in lecture and now we are going to put all of the things we have learned into one simple bit of code.

By now you know what `mystery_function()` does. You will now use that in a super special way to analyze the text that it uses.

We're going to use lists, dictionaries and files to produce some very interesting code.

§ Task: 3.1 Build a map (dictionary) of all the words in `mystery_function`.

Here we will show how easy it is and how flexible the use of dictionaries can truly be.

We will build a dictionary `d` which will have 13 *keys*: the numbers 1 to 13.

The *value* of the *keys* will be a list of the *unique* words which have are the length of the key (remember *key* is a number).

For example, the dictionary *key* `3` indicates words of length 3. The *value* of that key will be a list of the words from `mystery_function()` which are length 3.

Consider the words of length 3: `see, saw, red, has, for, one`, then in the dictionary `d` key `3` will be :

```
# an example dictionary, they key is the word length,
# the value is the list of words of that length (e.g. 3)

{
  3: ['see', 'saw', 'red', 'has', 'for', 'one']
}
```

You will write a cell which processes every word from `mystery_function`, uses your `find_word` function and processes all output from length 1 to 13.

Your final dictionary `d` will have keys from 1 to 13 representing words of length 1 to length 13 and each key's value will be the list of *unique* words of that length.

Here is an example:

```
{
  1: ['a', 'i'], # words of length 1
  2: ['me', 'to', 'us'], # words of length 2
  3: ['you', 'the', 'may', 'end'], # words of length 3
  4: ['them', 'they', 'four', 'shut'],
  5: ['saver', 'thumb', 'store'] # words of length 5
}
```

§ Task: 3.2 Write a function to generate fake sentences based on a template.

Now that we have the dictionary (map) of all the lengths from 1 to 13 and the words from `mystery_function` which are binned into lists of the same lengths, we are going to do one more fun thing.

You are going to write a function called `generate_sentence()` which will take a list of numbers and a dictionary (map) that maps word lengths to words of that length and returns a string which is a sentence composed of random words from the dictionary whose keys matches the numbers in the input list *in order*.

For example, if the input list contains the five numbers: `[3, 4, 5, 1, 3]`, then the output would be 5 words where the first is 3 letters, second 4, third 5, fourth 1 and last 3. So a string like "the bird sings a ton" would satify the requirements.

To be clear, the input string to your `generate_sentence()` function will be a list of numbers which represent the word lengths of the return string which is composed of random words of those specified lengths in the order that they appear.

Here are more formal details:

function **NAME**: `generate_sentence`

function **INPUT**:

- `size_lst` → a list of numbers (integer word lengths, etc),
- `w_map` → a dictionary of word lengths (keys) and list of corresponding words of that length (values).

function **OUTPUT**:

- a string containing random words of the size of each number in `size_lst`

Code Example:

```
w_map = your_code_to_make_a_map(mystery_list())
size_lst = [3, 4, 5, 1, 3]
generate_sentence(size_lst, w_map) # return a sentence like "the bird sings a ton"
```

Output:

```
"the bird sings a ton" # assuming these words are in w_map
```

Don't overthink this – your solution will be a very simple.

Here are some hints:

1. you should `import random` to be able to use random numbers,
2. `random.randint(1,5)` returns a number between 1 and 5,
3. using the fact from (2) you can use that number as a key to the dictionary `w_map`,
4. suppose `w_map = { 3: ['the', 'for'] }`; study and execute the code:

   ```
   w_len = 3

   # returns a random number between 1 and the length of w_map[w_len]
   k     = random.randint(1, len(w_map[w_len]))
   word  = w_map[w_len][k] # returns a random word from w_map[w_len]
   ```

   once you understand what is going on in this code you will have all the ingredients,
5. put this all together and return the string.

§ Task: 3.3 Use a template to store fake sentence fragments to a file.

In the starter notebook, there is a list called `sentence_template` which looks like this:

```
sentence_template =  [
    [1, 5, 2, 7, 2, 8],
    [3, 6, 3, 5, 4, 9],
    [3, 3, 6, 5, 3, 4],
    [4, 4, 8, 4, 2, 5],
    [4, 8, 4, 4, 1, 6]
]
```

Write the `for` loop over the `sentence_template`, run `generate_sentence()` from your prior solution and store **all** 5 sentences in a file with one sentence per line (in the file).

You sentence fragment file should be named `hw2_task3-3.txt` and here is an example:

```
i alone no trumpet at elements
him future but began just apartment
day one moment power now warm
that club cheering gift no among
went virtuoso 1965 club a people
```