Problem 1: Short Answer (20 points)
Put your answers into the boxes provided. We will not accept work that is outside the boxes.

A. Write the output for the following code:

```
grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(grid[2])
print(grid[1][1])
```

```
[7, 8, 9]
5
```

(5 points)

B. Write the output for the following code:

```
def f(s):
    return s.lower()

tup = ("Cat", "dog", "Bird", "FISH", "hamster")
result = sorted(tup, key=f)
print(result)
```

```
['Bird', 'Cat', 'dog', 'FISH', 'hamster']
```

(5 points)

C. Write the output for the following code:

```
data = {
      "students": [
            ("Alice", [85, 95]),
            ("Bob", [78, 88]),
            ("Charlie", [90, 95])
      ],
      "course": "CS106A"
}

x = data["students"][1][0]
y = data["course"]
z = (data["students"][0][1][0] + data["students"][0][1][1]) / 2

print(x)
print(y)
print(z)
```

```
Bob
CS106A
90.0
```

(5 points)

D. Write the output for the following code:

```
s = "poets describe dreams with words"
lst = []
w = ""
for ch in s:
      if ch == " ":
            lst.insert(0, w)
            w = ""
      else:
            w += ch
lst.insert(0, w)

s2 = ""
for element in lst:
      s2 += element + " "
print(s2)
```

```
words with dreams describe poets
```

(5 points)

Problem 2: Lists of Lists (20 points)

2A. Implement a function named `str_and_length(lst)` that takes a list of strings as its argument and returns a list of two-item lists. Each sublist should contain a string from the original list as its first element and the length of that string as its second element. (10 points)

Example 1:

```
lst = ["a", "bc", "def", "ghij"]
new_lst = str_and_length(lst)
print(new_lst)
```

output:
```
[['a', 1], ['bc', 2], ['def', 3], ['ghij', 4]]
```

Example 2:

```
lst = ['xyz', '', 'x', 'wxyz']
new_lst = str_and_length(lst)
print(new_lst)
```

output:
```
[['xyz', 3], ['', 0], ['x', 1], ['wxyz', 4]]
```

Example 3:

```
lst = []
new_lst = str_and_length(lst)
print(new_lst)
```

output:
```
[]
```

Your code:

```
def str_and_length(lst):
    new_lst = []
    for str in lst:
        new_lst.append([str, len(str)])
    return new_lst
```

2B: Use your `str_and_length()` function from part 2A in a new function, `count_map()` that returns a dictionary where the keys are the lengths, and the values are lists of all the strings that have that length. For example:

```
lst = ["a", "bc", "de", "f"]
new_lst = str_and_length(lst)
# new_lst contains [['a', 1], ['bc', 2], ['de', 2], ['f', 1]]
count_map(new_lst) # returns {1: ['a', 'f'], 2: ['bc', 'de']}
```

```
def count_map(lst):
    """
    lst will be in the form: [[str1, len1], [str2, len2],...]
    """
    count_dict = {}
    for str_and_len in lst:
        val = str_and_len[0]
        key = str_and_len[1]
        if key not in count_dict:
            count_dict[key] = []
        count_dict[key].append(val)
    return count_dict
```

Problem 3: Strings (20 points)

LOL-speak is a stylized form of English that has playful translations for certain words, and was popular in the early 2000s on internet forums. Here are some example translations:

hello, how are you today?	oh hai, how iz u todayz?
my cat is very fluffy.	mah kitteh iz sooo floofy.
do you have any snacks?	do u haz any snackz?

Write a function, `lol_speakify(s)` that takes a string parameter and uses a dictionary of words to their LOL-speak translations to convert the string to lol-chat. The string will have words separated by spaces, but the words can have punctuation on the end. You should keep punctuation in your final translations. You can assume you have the following function that removes punctuation and whitespace from a string:

```
def remove_punct(s):
    # removes all punctuation and spaces from a string
    result = ''
    for ch in s:
        if ch.isalnum(): # alphanumeric
            result += ch
    return result
```

For example:
```
>>> remove_punct('Hello!’)
'Hello'
>>> remove_punct('...cheeseburger...')
'cheeseburger'
```

You can also use the Python function `s.replace(s1, s2)`, which replaces all instances of `s1` with `s2` in a string, `s`. For example:

```
>>> s = "hello!!!!"
>>> new_s = s.replace('hello', 'hi')
>>> new_s
'hi!!!!'
```

Rules for your function:

1. You may not loop through the dictionary. Instead, you must process the string itself (you may want to use the string’s `split()` function
2. You can assume that the string parameter will be lowercase.
3. You may have an extra space on the end of the string you return.

Examples:

```
>>> lol_speakify("hello! can i please have your cheesburger? my tummy wants
food! thanks!")
'oh hai! can i plz haz ur cheesburger? mah tummy wants nomz! fank yoo! '
>>> lol_speakify("hello, how are you today?")
'oh hai, how r u today? '
>>> lol_speakify("my cat is fluffy. your cat is fluffy too.")
'mah kitteh is floofy. ur kitteh is floofy too. '
>>>
```

put ur code on the next page. fank you!

```
def lol_speakify(s):
    lol_dict = {
        "hello": "oh hai",
        "yes": "ya",
        "no": "nu",
        "please": "plz",
        "thanks": "fank yoo",
        "your": "ur",
        "you": "u",
        "are": "r",
        "have": "haz",
        "my": "mah",
        "food": "nomz",
        "cat" : "kitteh",
        "fluffy": "floofy"
    }

    # your code here!

    result = ''
    words = s.split()
    for word in words:
        stripped = remove_punct(word)
        if stripped in lol_dict:
            result += word.replace(stripped, lol_dict[stripped]) +  ' '
        else:
            result += word + ' '
    return result
```

Problem 4: File Processing and Nested Structures (20 points)

For assignment five, you wrote an Adventure game. The original text-based adventure games were often presented in a labyrinth of rooms with the ability to walk from room to room in the labyrinth. The rooms could contain objects to pick up, or monsters to fight. We can model a labyrinth as a grid, where each room in the grid can hold items or monsters. Here is an example 3 row x 2 column labyrinth:

`items: sword, potion`
`monsters: orc`
	`items: map`
	`monsters: none`

`items: medicine`
`monsters: none`
	`items: none`
	`monsters: none`

`items: fireball,`
`shield`
`monsters: snake, bunny`
	`items: none`
	`monsters: troll`

You are given a file that describes a labyrinth as follows:

- the first line of the file is the number of rows in grid
- the second line of the file is the number of columns in the grid
- the remaining lines represent the items and monsters for each grid location, row-by-row and column by column. First there will be a line of items for a grid location, then the next line will contain the monsters for that location. If there are multiple items or monsters in a location, they will be space separated.
- If there are no items or monsters in a location, the line will be blank

Here is the file for the grid above:

`3`	←	`First line: Number of rows`
`2`	←	`Second line: Number of columns`
`sword potion`	←	`grid[0][0] items`
`orc`	←	`grid[0][0] monsters`
`map`	←	`grid[0][1] items`
	←	`grid[0][1] monsters`
`medicine`	←	`grid[1][0] items`
	←	`grid[1][0] monsters`
	←	`grid[1][1] items`
	←	`grid[1][1] monsters`
`fireball shield`	←	`grid[2][0] items`
`snake bunny`	←	`grid[2][0] monsters`
	←	`grid[2][1] items`
`troll`	←	`grid[2][1] monsters`

For this problem, write the function `read_labryinth_from_file(filename)` that creates and returns a list of lists for the grid, with each grid location containing a dictionary with the keys "`items`" and "`monsters`" and lists of the items and monsters for each location.

Notes:

- You need to use the row count and column count to read the rest of the file
- You should populate your lists and dictionaries as you read the remaining data

The next page shows the nested list created from the file above.

The nested list created from the file on the previous page, which contains one sublist per row all contained in one overall list:

```
[
    [
        {'items': ['sword', 'potion'], 'monsters': ['orc']},
        {'items': ['map'], 'monsters': []}
    ],
    [
        {'items': ['medicine'], 'monsters': []},
        {'items': [], 'monsters': []}
    ],
    [
        {'items': ['fireball', 'shield'], 'monsters': ['snake', 'bunny']},
        {'items': [], 'monsters': ['troll']}
    ]
]
```

Your code:

```
def read_labryinth_from_file(filename):
    labryinth = []
    with open(filename, 'r') as f:
        num_rows = int(f.readline())
        num_cols = int(f.readline())
        for row in range(num_rows):
            new_row = []
            for col in range(num_cols):
                items = f.readline().split()
                monsters = f.readline().split()
                new_row.append({'items': items, 'monsters': monsters})
            labryinth.append(new_row)
    return labryinth
```

Problem 5: Sorting (20 points)

For this problem, you will be given a list of historical speeches as a list of tuples, in the form of `(speaker, speech_title, year_delivered, number_of_words_in_the_speech)`, e.g.,

```
speeches = [
    ("Martin Luther King Jr.", "I Have a Dream", 1963, 1664),
    ("Abraham Lincoln", "Gettysburg Address", 1863, 272),
    ("Hillary Clinton", "Women's Rights Are Human Rights", 1995, 2631),
    ("Barack Obama", "Yes We Can", 2008, 2151),
    ("Eleanor Roosevelt", "The Struggle for Human Rights", 1948, 1865)
]
```

The goal will be to sort the speeches by the **average words per year since the speech was given**, and to display them from **highest to lowest** average words per year.

Let's break it down as follows.

A. Write the function **avg_words_per_year(speech)** function which accepts a tuple and returns the average number of words since the current year (2025, in our case), rounded to one decimal place. In other words, you want to calculate the following:

```
average_words_per_year = word_count / (CURRENT_YEAR - year)
```

The round function is defined as follows, which rounds a `number` to `ndigits`:

```
round(number, ndigits)
```

Example:

```
>>> round(1.777, 1)
1.8
```

Write the function below:

```
CURRENT_YEAR = 2025
def avg_words_per_year(speech):
    # example input:
    # ("Martin Luther King Jr.", "I Have a Dream", 1963, 1664)
    # YOUR CODE HERE:
    average_words_per_year = round(speech[3] / (CURRENT_YEAR - speech[2]),1 :)
    return average_words_per_year
```

B. Next, use the `sorted()` function to create a sorted list from the following speeches list of tuples that is in order from highest to lowest average number of words per year:

```
speeches = [
    ("Martin Luther King Jr.", "I Have a Dream", 1963, 1664),
    ("Abraham Lincoln", "Gettysburg Address", 1863, 272),
    ("Hillary Clinton", "Women's Rights Are Human Rights", 1995, 2631),
    ("Barack Obama", "Yes We Can", 2008, 2151),
    ("Eleanor Roosevelt", "The Struggle for Human Rights", 1948, 1865)
]

sorted_speeches = sorted(speeches, key=avg_words_per_year, reverse=True)
```

C. Finally, write a `for` loop that prints each speech from `sorted_speeches` in the following form:

```
Barack Obama: 'Yes We Can' (2008) - 126.53 words per year
Hillary Clinton: 'Women's Rights Are Human Rights' (1995) - 87.7 words per year
etc.
```

Write your code below:

```
for speech in sorted_speeches:
    # YOUR CODE HERE:
    for speech in sorted_speeches:
        awpy = avg_words_per_year(speech)
        print(f"{speech[0]}: '{speech[1]}' ({speech[2]}) - {awpy} words per
year")
```

Problem 6: Classes (20 points)

Implement a class **`Library`** with the following methods:

- **`init(self)`** - initializes a new **`Library`** instance and any member variables
- **`return_book(self, title)`** - returns a book to the library with the given title
- **`check_out(self, title)`** - removes a book from the library with the given title, or prints 'No book with that title' if that book isn't found
- **`get_catalog(self)`** - returns a list of all books currently in the library, sorted alphabetically

Here's how your class might be used from another program.

```
from library import Library


def main():
    green_library = Library()
    green_library.return_book('Karel the Robot Learns Python')
    green_library.return_book('The Giving Tree')
    green_library.return_book('Design as Art’)

    # prints ['Design as Art', 'Karel the Robot Learns Python', 'The Giving Tree']
    print(green_library.get_catalog()
    green_library.check_out('The Giving Tree’)

    # prints No book with that title
    green_library.check_out(‘Annihilation’)

    # prints ['Design as Art', 'Karel the Robot Learns Python']
    print(green_library.get_catalog())


if __name__ == '__main__':
    main()
```

On the following page, please implement the `Library` class

```
class Library:

    def __init__(self):
        """
        Creates a new instance of the Library class
        """
        # instance variable to keep track of the book titles
        self.library = []


    def return_book(self, title):
        """
        Returns a book with the given title to the library
        """
        self.library.append(title)


    def check_out(self, title):
        """
        Removes a book with the given title from the library, or prints
        'No book with that title' if that book isn't found
        “”"
        if title in self.library:
            self.library.remove(title)
        else:
            print(“No book with that title”)


    def get_catalog(self):
        """
        Returns a list of all books currently in the library, sorted
        alphabetically
        """
        return sorted(self.library)
```