Final Project

Learning Objective

Demonstrate knowledge of the Python programming concepts learned throughout the semester. **You are only allowed to use Python concepts taught in this course.**

Project Description

Write a program that allows the user to play a game of Connections. The user is given 16 words that need to be connected into 4 groups of 4. Sample output is shown on pages 3 – 8, and steps are described on pages 9 – 20. Read page 21 regarding grading. The extra credit opportunity is described on pages 22 – 23.

Read a CSV (comma-separated values) file with puzzles to create **a list of dictionaries. Each dictionary holds the information for one puzzle where the keys are strings and the values are strings**. You are provided a CSV file named **puzzles.csv**.

- Each line represents one row in a table, and commas separate each column.
- The first line in the CSV file represents the header (column labels) that contains the keys for the dictionaries. Each subsequent row represents a puzzle (values for the dictionary).

CSV File (header row in light cardinal, each row in an alternating yellow & grey)

```
num,date,color1,difficulty1,connection1,word11,word12,word13,word14,color2,difficulty2,connection2,
word21,word22,word23,word24,color3,difficulty3,connection3,word31,word32,word33,word34,color4,
difficulty4,connection4,word41,word42,word43,word44
1,2023 Jun 12,YELLOW,1,WET WEATHER,HAIL,RAIN,SLEET,SNOW,GREEN,2,NBA TEAMS,BUCKS,HEAT,JAZZ,NETS,
BLUE,3,KEYBOARD KEYS,OPTION,RETURN,SHIFT,TAB,PURPLE,4,PALINDROMES,KAYAK,LEVEL,MOM,RACE CAR
2,2023 Jun 13,YELLOW,1,FOOTWEAR,BOOT,LOAFER,PUMP,SNEAKER,GREEN,2,UNITS OF LENGTH,FOOT,LEAGUE,MILE,
YARD,BLUE,3,MAGAZINES,ESSENCE,PEOPLE,TIME,US,PURPLE,4,LETTER HOMOPHONES,ARE,QUEUE,SEA,WHY
...
507,2024 Oct 30,YELLOW,1,UPSWING,BOOM,RISE,SPIKE,SURGE,GREEN,2,THINGS WITH WHEELS,DOLLY,ROLLERBLADE,
SKATEBOARD,WAGON,BLUE,3,KINDS OF TAPE,DUCT,ELECTRICAL,GAFFER,PACKING,PURPLE,4,RETAIL CHAINS WITH A
LETTER CHANGED,BEST BOY,IDEA,KRONER,STABLES
508,2024 Oct 31,YELLOW,1,TERMS OF ENDEARMENT,DARLING,LOVE,PUMPKIN,TREASURE,GREEN,2,THINGS YOU CAN DO
WITH YOUR EYELIDS,BAT,BLINK,FLUTTER,WINK,BLUE,3,SPORTS CARS,DIABLO,MUSTANG,SPIDER,VIPER,PURPLE,4,___
HUNT,EGG,JOB,SCAVENGER,WITCH
```

CSV Data in a Table (Partial, first 3 rows and last 2 rows, first 9 columns)

num	date	color1	difficulty1	connection1	word11	word12	word13	word14	...
1	2023 Jun 12	YELLOW	1	WET WEATHER	HAIL	RAIN	SLEET	SNOW	
2	2023 Jun 13	YELLOW	1	FOOTWEAR	BOOT	LOAFER	PUMP	SNEAKER	
...									
507	2024 Oct 30	YELLOW	1	UPSWING	BOOM	RISE	SPIKE	SURGE	
508	2024 Oct 31	YELLOW	1	TERMS OF ENDEARMENT	DARLING	LOVE	PUMPKIN	TREASURE	

NYT Connections data from Word tips.

The rules displayed to the user are located in a text file named **rules.txt**, which contains:

```
How to Play
Find groups of four items that share something in common.
* Enter four items to check if your guess is correct.
* Find the groups without making 4 mistakes.
Category Examples:
* FISH: BASS, FLOUNDER, SALMON, TROUT
* FIRE ___: ANT, DRILL, ISLAND, OPAL
Categories will always be more specific than "5-LETTER-WORDS", "NAMES", or "VERBS".
Each puzzle has exactly one solution. Watch out for words that seems to belong to multiple
categories.
Each group is assigned a color, which will be revealed as you solve:
YELLOW: Straightforward
GREEN:    |
BLUE:     V
PURPLE: Tricky
```

In order to display the game board with colors and items, we created some functions in the **pretty_print.py** Python file. You will need to call the **displayGrid()** function:

```
# Function: displayGrid
# Parameter 1: puzzleDict is a dictionary representing one puzzle
# Parameter 2: foundGroupList is a list of integers with the difficulty numbers
# of the groups that have been found and has a default value of [1, 2, 3, 4]
# Parameter 3: mistakes is an integer with the number of mistakes used to display
# the mistakes remaining message and has a default value of -1 (do not display message)
```

You will create three Python files: **helper.py**, **interface.py**, and **main_*last_first*.py**.

- In the helper.py file, create the following functions to read the CSV file, create a list of dictionaries, and perform tasks: **createPuzzleList()**, **getPuzzle()**, **getColor()**, **getConnection()**, **getWordsInGroup()**, and **getUnconnectedWords()**.
- In the interface.py file, create the following functions to interact with the user: **displayRules()**, **isValidNumber()**, **pickPuzzle()**, **getGuessList()**, **checkConnection()**, **savePuzzle()**, and **playGame()**.
- In the main_*last_first*.py file, create the **main()** function which calls the functions in the other files.

Requirements

Use Python concepts from this semester's course materials including slide decks, videos, assignments, and labs. They are all available on Brightspace. If you need help, go to the office hours of any instructor or learning assistant (LA) or post on Piazza.

Do NOT import and use the CSV or other modules to process the CSV file.
Do NOT collaborate, share code, or exchange solutions with others.
Do NOT look for help on other sites such as Chegg, Course Hero, and Stack Overflow.
Do NOT use an AI program such as ChatGPT to help complete this project.

Sample Output #1 (user input shown in green)

```
Connections!
Group words that share a common thread.

How to Play
Find groups of four items that share something in common.
* Enter four items to check if your guess is correct.
* Find the groups without making 4 mistakes.
Category Examples:
* FISH: BASS, FLOUNDER, SALMON, TROUT
* FIRE ___: ANT, DRILL, ISLAND, OPAL
Categories will always be more specific than "5-LETTER-WORDS", "NAMES", or
"VERBS".
Each puzzle has exactly one solution. Watch out for words that seems to
belong to multiple categories.
Each group is assigned a color, which will be revealed as you solve:
YELLOW: Straightforward
GREEN:    |
BLUE:     V
PURPLE: Tricky

Enter a number (1-508): 1

        Create four groups of four!
SNOW        TAB         RACE CAR    BUCKS
JAZZ        RAIN        RETURN      MOM
OPTION      NETS        HEAT        SLEET
LEVEL       HAIL        KAYAK       SHIFT
        Mistakes remaining: * * * *

Enter four items for your guess
Item #1:    return
Item #2: tab
Item #3: OPTION
Item #4:   Shift

        Create four groups of four!
              KEYBOARD KEYS
         OPTION, RETURN, SHIFT, TAB
BUCKS       SNOW        LEVEL       JAZZ
NETS        MOM         HEAT        HAIL
KAYAK       RAIN        RACE CAR    SLEET
        Mistakes remaining: * * * *

Enter four items for your guess
```

```
Item #1: snow
Item #2: HAIL
Item #3: rain
Item #4: snow
Item #4: slet
Item #4: sleet

        Create four groups of four!
              KEYBOARD KEYS
         OPTION, RETURN, SHIFT, TAB
               WET WEATHER
          HAIL, RAIN, SLEET, SNOW
NETS       LEVEL      KAYAK      MOM
HEAT       RACE CAR   BUCKS      JAZZ
        Mistakes remaining: * * * *

Enter four items for your guess
Item #1: jazz
Item #2: level
Item #3: kayak
Item #4: mom

        Create four groups of four!
              KEYBOARD KEYS
         OPTION, RETURN, SHIFT, TAB
               WET WEATHER
          HAIL, RAIN, SLEET, SNOW
HEAT       RACE CAR   JAZZ       KAYAK
MOM        NETS       LEVEL      BUCKS
         Mistakes remaining: * * *

Enter four items for your guess
Item #1: heat
Item #2: nets
Item #3: jazz
Item #4: bucks

        Create four groups of four!
              KEYBOARD KEYS
         OPTION, RETURN, SHIFT, TAB
               WET WEATHER
          HAIL, RAIN, SLEET, SNOW
                NBA TEAMS
          BUCKS, HEAT, JAZZ, NETS
MOM        KAYAK      LEVEL      RACE CAR
         Mistakes remaining: * * *
```

```
Enter four items for your guess
Item #1: mom
Item #2: kayak
Item #3: level
Item #4: race
Item #4: race car

        Create four groups of four!
```

KEYBOARD KEYS OPTION, RETURN, SHIFT, TAB
WET WEATHER HAIL, RAIN, SLEET, SNOW
NBA TEAMS BUCKS, HEAT, JAZZ, NETS
PALINDROMES KAYAK, LEVEL, MOM, RACE CAR

```
Congratulations!
Puzzle from 2023 Jun 12 has been saved to puzzle1.txt
```

Output to puzzle1.txt File:

```
2023 Jun 12
BLUE: KEYBOARD KEYS ['OPTION', 'RETURN', 'SHIFT', 'TAB']
YELLOW: WET WEATHER ['HAIL', 'RAIN', 'SLEET', 'SNOW']
GREEN: NBA TEAMS ['BUCKS', 'HEAT', 'JAZZ', 'NETS']
PURPLE: PALINDROMES ['KAYAK', 'LEVEL', 'MOM', 'RACE CAR']
```

Sample Output #2 (user input shown in green)

```
Connections!
Group words that share a common thread.

How to Play
Find groups of four items that share something in common.
* Enter four items to check if your guess is correct.
* Find the groups without making 4 mistakes.
Category Examples:
* FISH: BASS, FLOUNDER, SALMON, TROUT
* FIRE ___: ANT, DRILL, ISLAND, OPAL
Categories will always be more specific than "5-LETTER-WORDS", "NAMES", or
"VERBS".
Each puzzle has exactly one solution. Watch out for words that seems to
belong to multiple categories.
Each group is assigned a color, which will be revealed as you solve:
YELLOW: Straightforward
GREEN:    |
BLUE:     V
PURPLE: Tricky

Enter a puzzle number (1-508): 250

              Create four groups of four!
UNION         SHUTTLECOCK   MOCCASIN      BASIC
LIGHTNING     BOA           CROC          RUBY
HEADDRESS     LOAFER        PYTHON        INSPIRATION
SLIPPER       PILLOW        JAVA          COBRA
              Mistakes remaining: * * * *

Enter four items for your guess
Item #1: Python
Item #2: JAVA
Item #3:   BASIC
Item #4: ruby

              Create four groups of four!
                 PROGRAMMING LANGUAGES
               BASIC, JAVA, PYTHON, RUBY
HEADDRESS     COBRA         LOAFER        BOA
SHUTTLECOCK   CROC          PILLOW        SLIPPER
LIGHTNING     MOCCASIN      INSPIRATION   UNION
              Mistakes remaining: * * * *

Enter four items for your guess
```

```
Item #1: slipper
Item #2: moccasin
Item #3: pillow
Item #4: loafer

              Create four groups of four!
                 PROGRAMMING LANGUAGES
                BASIC, JAVA, PYTHON, RUBY
INSPIRATION   COBRA         SHUTTLECOCK   BOA
PILLOW        HEADDRESS     CROC          LIGHTNING
UNION         LOAFER        SLIPPER       MOCCASIN
               Mistakes remaining: * * *

Enter four items for your guess
Item #1: boa
Item #2: cobra
Item #3: croc
Item #4: shuttlecock

              Create four groups of four!
                 PROGRAMMING LANGUAGES
                BASIC, JAVA, PYTHON, RUBY
SHUTTLECOCK   PILLOW        SLIPPER       HEADDRESS
CROC          INSPIRATION   LOAFER        COBRA
UNION         MOCCASIN      BOA           LIGHTNING
                Mistakes remaining: * *

Enter four items for your guess
Item #1: headdress
Item #2: shuttlecock
Item #3: pillow
Item #4: slipper

              Create four groups of four!
                 PROGRAMMING LANGUAGES
                BASIC, JAVA, PYTHON, RUBY
INSPIRATION   BOA           MOCCASIN      HEADDRESS
COBRA         PILLOW        UNION         LOAFER
LIGHTNING     CROC          SLIPPER       SHUTTLECOCK
                 Mistakes remaining: *

Enter four items for your guess
Item #1: loafer
Item #2: slipper
Item #3: moccasin
Item #4: croc
```

```
              Create four groups of four!
                  PROGRAMMING LANGUAGES
                 BASIC, JAVA, PYTHON, RUBY
                        COMFY SHOES
                CROC, LOAFER, MOCCASIN, SLIPPER
COBRA           SHUTTLECOCK    INSPIRATION    UNION
PILLOW          LIGHTNING      BOA            HEADDRESS
                   Mistakes remaining: *

Enter four items for your guess
Item #1: boa
Item #2: cobra
Item #3: shuttlecock
Item #4: lightning

              Create four groups of four!
                        COMFY SHOES
                CROC, LOAFER, MOCCASIN, SLIPPER
                  THINGS MADE WITH FEATHERS
              BOA, HEADDRESS, PILLOW, SHUTTLECOCK
                  PROGRAMMING LANGUAGES
                 BASIC, JAVA, PYTHON, RUBY
                  THINGS THAT CAN STRIKE
              COBRA, INSPIRATION, LIGHTNING, UNION

Better luck next time.
Puzzle from 2024 Feb 16 has been saved to puzzle250.txt
```

Output to puzzle250.txt File:

```
2024 Feb 16
YELLOW: COMFY SHOES ['CROC', 'LOAFER', 'MOCCASIN', 'SLIPPER']
GREEN: THINGS MADE WITH FEATHERS ['BOA', 'HEADDRESS', 'PILLOW', 'SHUTTLECOCK']
BLUE: PROGRAMMING LANGUAGES ['BASIC', 'JAVA', 'PYTHON', 'RUBY']
PURPLE: THINGS THAT CAN STRIKE ['COBRA', 'INSPIRATION', 'LIGHTNING', 'UNION']
```

Steps

1. In PyCharm (Community Edition), open your existing ITP115 project.
2. Create a new directory called **project_*last_first*** where *last* is your last/family name and *first* is your preferred first name. Use all lowercase letters. You will have points deducted if this is not correct.
3. In the directory, you will create **multiple Python files**. At the top of **each file**, put comments in the following format and replace the name, email, section, and filename with your actual information:

```
# ITP 115, Fall 2024
# Final Project
# Name: First Last
# Email: username@usc.edu
# Section: number or nickname
# Filename: filename.py (update accordingly)
# Description: (you fill in)
```

4. Put the **puzzles.csv** file in your project_*last_first* directory.
   - Download the file from Brightspace under the item for the Final Project.
   - Drag it onto your project_*last_first* directory in PyCharm.
   - If needed, use the Refactor → Rename tool in PyCharm to change the filename to puzzles.csv.
   - Do not open it in Excel or Google Sheets. It can change the encoding. If you did, delete the previous version and download it again from Brightspace.
5. Put the **rules.txt** file in your project_*last_first* directory.
   - Download the file from Brightspace by right-clicking on it and selecting the "Save Link As..." option.
   - Navigate to your project_*last_first* folder.
   - Click the Save button.
6. Put the **pretty_print.py** file in your project_*last_first* directory.
   - Download the file from Brightspace by right-clicking on it and selecting the "Save Link As..." option.
   - Navigate to your project_*last_first* folder.
   - Click the Save button.

7. Create a Python file entitled **helper.py**. In this file, define the following functions that will be called from other Python files. At the top of this file, add the comment block that must include your name.
   - Define the **createPuzzleList()** function in the helper.py file.

     Function: createPuzzleList
     Parameter: filenameStr is a string with name of the CSV file to read and it has a default value of "puzzles.csv"
     Return value: a list of dictionaries where each dictionary represents one puzzle
     This function reads the CSV file and creates a list of puzzles which is the main data structure used in this program.

     ◊ Each puzzle is represented with a dictionary. The keys are strings from the header row (first row in the CSV file):

     ```
     "num", "date", "color1", "difficulty1", "connection1", "word11", "word12",
     "word13", "word14", "color2", "difficulty2", "connection2", "word21", "word22",
     "word23", "word24", "color3", "difficulty3", "connection3", "word31", "word32",
     "word33", "word34", "color4", "difficulty4", "connection4", "word41", "word42",
     "word43", "word44"
     ```

     ◊ Each row for the remaining rows of the CSV file has information for one puzzle. For example, the values for the first dictionary are in the second row in the CSV file:

     ```
     "1", "2023 Jun 12", "YELLOW", "1", "WET WEATHER", "HAIL", "RAIN", "SLEET",
     "SNOW", "GREEN", "2", "NBA TEAM", "BUCKS", "HEAT", "JAZZ", "NETS", "BLUE", "3",
     "KEYBOARD KEYS", "OPTION", "RETURN", "SHIFT", "TAB", "PURPLE", "4",
     "PALINDROMES", "KAYAK", "LEVEL", "MOM", "RACE CAR"
     ```

     ◊ Create a variable to hold a list of puzzles. Open a file for reading, which creates a file variable. Read the first line using the file.readline() method, remove the new line ("\n"), and use the appropriate string method to split the string into a list (keys for each dictionary).

     ◊ Loop through the rest of the file. In the outer loop, create an empty dictionary, get the information for one puzzle, remove the new line character, and split it into a list (these are the values for the dictionary).

     ◊ Now you have one list with keys and another list with values where both lists are the same length. Use a nested (or inner) loop based to put the data in the dictionary. The data in the two lists is matched by an index.

     ◊ After the nested loop, add the dictionary to the list.

     ◊ After the outer loop, close the file and return the list.

     ◊ To test this function, call it and capture the return value in a variable. You could print the variable, but there are 508 dictionaries. We suggest using slicing to only print the first 3 dictionaries. You can also print the length of the list to verify that it

contains 508 items. If you encounter an error named UnicodeDecodeError, add an argument for the encoding parameter (encoding="UTF-8") when calling the open() function.

- ◊ This function will be called once in the main() function in the main_*last_first*.py file. The return value will be captured in a variable and used as an argument when calling other functions that have puzzleList as a parameter.

- Define the **getPuzzle()** function in the helper.py file.

  Function: getPuzzle
  Parameter 1: puzzleList is a list of dictionaries where each dictionary represents a puzzle
  Parameter 2: numStr is a string which is one of the valid values in puzzleList for the "num" key
  Return value: a dictionary representing one puzzle where the value for the "num" key matches the numStr parameter
  This function uses the two parameters to return one dictionary which represents one puzzle.

  - ◊ This function will have the following structure: Loop through the puzzleList parameter. For each dictionary, make sure the "num" key is in the dictionary. If so, get the value based on the "num" key and compare it to the numStr parameter. If they match, return that dictionary.
  - ◊ Return an empty dictionary if the numStr is not valid or the puzzleList is empty.
  - ◊ You are NOT allowed to use a set or tuple.
  - ◊ This function will be called in a function in the interface.py file.

- Define the **getColor()** function in the helper.py file.

  Function: getColor
  Parameter 1: puzzleDict is a dictionary that represents one puzzle
  Parameter 2: difficultyNum is an integer which is one of the difficulty levels (1 – 4)
  Return value: a string with the color that corresponds to the difficulty level
  This function returns a string ("YELLOW", "GREEN", "BLUE", "PURPLE") based on the difficultyNum parameter.

  - ◊ The keys in the dictionary are "color1", "color2", "color3", and "color4". Notice the key is the string "color" with the difficulty level added to it.
  - ◊ This function will have the following structure: Create a variable for the key ("color" and difficultyNum parameter). Make sure the key is in the puzzleDict. If so,

get the value from the puzzleDict parameter and return it. If the key is not in the dictionary, return the string "GREY".

- ◊ This function will be called in the pretty_print.py file, which is provided for you, and in the function savePuzzle(), which you will define in the interface.py file.

- Define the **getConnection()** function in the helper.py file.

  Function: getConnection
  Parameter 1: puzzleDict is a dictionary that represents one puzzle
  Parameter 2: difficultyNum is an integer which is one of the difficulty levels (1 – 4)
  Return value: a string with the connection that corresponds to the difficulty level
  This function returns the connection string based on the difficultyNum parameter.

  - ◊ The keys in the dictionary are "connection1", "connection2", "connection3", and "connection4".
  - ◊ This function will have the following structure: Create a variable for the key ("connection" and difficultyNum parameter). If the key is in the puzzleDict, get the value from the puzzleDict parameter and return it. If the key is not in the puzzleDict, return the string "CONNECTION ERROR".
  - ◊ This function will be called in the pretty_print.py file, which is provided for you, and in the function savePuzzle(), which you will define in the interface.py file.

- Define the **getWordsInGroup()** function in the helper.py file.

  Function: getWordsInGroup
  Parameter 1: puzzleDict is a dictionary that represents one puzzle
  Parameter 2: difficultyNum is an integer which is one of the difficulty levels (1 – 4)
  Return value: an alphabetically sorted list of strings which are the four items in a group based on the difficulty level

  - ◊ The keys in the dictionary start with the "word" followed by the difficultyNum and then the numbers 1 through 4. For example if the difficultyNum is 1, the keys are "word11", "word12", "word13", and "word14".
  - ◊ This function will have the following structure: Create a variable to hold the list of words. Loop through the numbers 1 to 4. Create a variable for the key using "word", the difficultyNum and the loop variable. If the key is in puzzleDict, get the value based on the key and append to the list. After the loop, sort the list and then return it.
  - ◊ This function will be called in the pretty_print.py file, which is provided for you, and in functions which you will define in the interface.py file.

- Define the **getUnconnectedWords()** function in the helper.py file.

  Function: getUnconnectedWords
  Parameter 1: puzzleDict is a dictionary that represents one puzzle
  Parameter 2: foundGroupList is a list of integers holding the difficulty levels (1 – 4) that have already been found
  Return value: a list of strings which are the items in the puzzle that have not been connected yet
  This function creates a list of the words that have not been connected.

  - ◊ This function will have the following structure: Create a variable to hold the list of words. Loop through the numbers 1 to 4. If the number is not in the foundGroupList, call the getWordsInGroup() function, and add the returned list to the list variable. After the loop, return the list.
  - ◊ You are not required to call the getWordsInGroup() function, but it is the easiest way to accomplish this.
  - ◊ To add one list to another list, you can use list concatenation. Alternatively, you can use a loop with the list.append() method. Do NOT use the list.extend() method.
  - ◊ To help understand this function, the foundGroupList parameter is created and updated in the playGame() function, which you will define in the interface.py file, to keep track of the groups of words that have been connected. We do not need to save the words, just the difficulty numbers (1 – 4). If the foundGroupList parameter is an empty list, this function will return a list with the 16 words. If the foundGroupList parameter is a list with only one number, the return list will have 12 words.
  - ◊ This function will be called in the pretty_print.py file, which is provided for you, and in the playGame() function you will define in the interface.py file.

8. Create a Python file entitled **interface.py**. In this file, define the following functions that interface with the user. Some functions will be called in function definitions inside this file and some functions will be called in the main() function in the main.py file. At the top of this file, add the comment block that must include your name.
   - Import the **helper** file to allow functions defined in this file to call functions in the helper.py file you created.
   - Import the **pretty_print** file to allow functions defined in this file to call functions in the pretty_print.py file you added to this project.
   - Create the **displayRules()** function in the interface.py file.

     Function: displayRules
     Parameter: textfileStr is a string with name of the text file to read and it has a default value of "rules.txt"
     Return value: None
     This function reads the text file and prints each line to the user.
     - ◊ Read each line and print to the user. Remember to close the file.
     - ◊ This function will be called once in the main() function in the main_*last_first*.py file.
   - Create the **isValidNumber()** function in the interface.py file.

     Function: isValidNumber
     Parameter 1: userStr is a string with input from the user
     Parameter 2: startNum is an integer which is the first number in the range of valid numbers
     Parameter 3: endNum is an integer which is the last number in the range of valid numbers (inclusive)
     Return value: a Boolean, True if the userStr parameter is a digit and a number in the range starting at startNum and ending at endNum (inclusive), otherwise False
     This function determines if the user entered a valid number.
     - ◊ Do not get user input in this function. Use the userStr parameter.
     - ◊ Return True if the userStr parameter only contains digits and is a number that is valid based on the startNum and endNum parameters. Otherwise, return False.
     - ◊ Remember that userStr is a string. If it only contains digits, then you can convert it to an int to test it with the startNum and endNum parameters.
     - ◊ This function will be called in the pickPuzzle() function in this file.

- Create the **pickPuzzle()** function in the interface.py file.

  Function: pickPuzzle
  Parameter: puzzleList is a list of dictionaries where each dictionary represents a puzzle
  Return value: a dictionary representing one puzzle based on the user's input for the puzzle number
  This function allows the user to enter input to pick the puzzle number.

  - ◊ Get the length of the puzzleList which will be the last valid puzzle number. The first valid puzzle number is 1.
  - ◊ Get input from the user based on the length of the puzzleList parameter (the cardinal text shows the text replace by a variable):

    ```
    Enter a puzzle number (1-508):
    ```

  - ◊ Use a loop to continue to ask the user for input until they enter valid input. Use the isValidNumber() function. Here is an example (user input in green text):

    ```
    Enter a puzzle number (1-508): number
    Enter a puzzle number (1-508): -5
    Enter a puzzle number (1-508): 295
    Enter a puzzle number (1-508): 1
    ```

  - ◊ Call the helper.getPuzzle() function and return the puzzle.
  - ◊ This function will be called in the main() function in the main_*last_first*.py file.

- Define the **getGuessList()** function in the interface.py file.

  Function: getGuessList
  Parameter: wordList is a list of strings which are the words from the puzzle that have not been connected yet
  Return value: a list of strings containing four items entered by the user that are sorted in alphabetical order
  This function creates a list of four strings with items that are from the wordList parameter.

  - ◊ Display the following message to the user:

    ```
    Enter four items for your guess
    ```

  - ◊ Create a variable for the return value which is the guess list.
  - ◊ Loop four times to get four valid items to add to the guess list.
  - ◊ In the loop, ask the user for input. Use a nested loop to continue getting input until the user enters a valid input. A valid input is in the wordlist parameter and has not been previously entered by the user (i.e., in the guess list). Allow the user to

enter extra spaces before and after the item. Allow the user to enter input using uppercase or lowercase letters. The words in the wordList are all uppercase.

- ◊ After the nested loop, add the user's input to the guess list.
- ◊ Here is an example where "RAIN", "SNOW", "HAIL", and "SLEET" are in the wordList parameter (user input is in green text and the cardinal text shows text replaced by a variable):

```
Item #1: raindrops
Item #1: rain
Item #2:    snow
Item #3: HAIL
Item #4: SNOW
Item #4: sleet
```

- ◊ After the outer loop, sort the list. Return the list.
- ◊ This function will be called in the playGame() function in this file.

- ○ Define the **checkConnection()** function in the interface.py file.

Function: checkConnection
Parameter 1: puzzleDict is a dictionary that represents one puzzle
Parameter 2: guessList is an alphabetically sorted list of strings with four items
Return value: an integer that is the difficulty number (1 – 4) of the connection group that matches guessList, otherwise 0 if guessList is not one of the connection groups
This function checks to see if the guessList is one of the connection groups in puzzleDict. If so, it returns the difficulty number. Otherwise, it returns 0.

- ◊ Create a variable for the return value. Use a loop and call the getWordsInGroup() function in the helper.py file.
- ◊ In the loop, use branching to determine whether the guessList parameter matches a connection group.
- ◊ This function will be called in the playGame() function in this file.

- Define the **savePuzzle()** function in the interface.py file.

  Function: savePuzzle
  Parameter 1: puzzleDict is a dictionary that represents one puzzle
  Parameter 2: groupsFound is a list of integers with the difficulty number and has a default value of [1, 2, 3, 4]
  Return value: None
  This function writes the puzzle to a file. The file's name is the text "puzzle" with the puzzle number and has a .txt file extension. Write the date in the file on its own line. Write each group on its own line in the following format:
  COLOR: CONNECTION ['WORD1', 'WORD2', 'WORD3', 'WORD4']

  - This function will have the following structure: Get the "num" from the puzzleDict to use for the filename. Create a variable for the filename that starts with "puzzle", includes the num, and has "txt" as the file extension. Open the file for writing. Get the "date" from the puzzleDict and write to the file using the print() function. Use a loop to repeat based on the groupsFound parameter. In the loop, call the getColor(), getConnection(), and getWordsInGroup() functions in the helper.py file. Write the appropriate data to the file. After the loop, close the file. Display a message to the user stating the puzzle date and the filename. Here is an example when the num is 250 (the cardinal text shows text replaced by variables):

```
Puzzle from 2024 Feb 16 has been saved to puzzle250.txt
```

  - Here is an example of the text in the puzzle250.txt file:

```
2024 Feb 16
YELLOW: COMFY SHOES ['CROC', 'LOAFER', 'MOCCASIN', 'SLIPPER']
GREEN: THINGS MADE WITH FEATHERS ['BOA', 'HEADDRESS', 'PILLOW', 'SHUTTLECOCK']
BLUE: PROGRAMMING LANGUAGES ['BASIC', 'JAVA', 'PYTHON', 'RUBY']
PURPLE: THINGS THAT CAN STRIKE ['COBRA', 'INSPIRATION', 'LIGHTNING', 'UNION']
```

  - This function will be called in the playGame() function in this file.

- Define the **playGame()** function in the interface.py file.

  Function: playGame
  Parameter: puzzleDict is a dictionary that represents one puzzle
  Return value: None
  This function allows the user to play a game by calling various functions. The game will display a 4x4 grid of words and ask users to enter 4 words (their guess) to make a connection. The game ends when the user has guessed all 4 connections or makes 4 mistakes.

  ◊ Create a variable to hold a list of integers which are the difficulty numbers (1 – 4) that the user has correctly guessed and initially set it to the empty list. Create a variable to hold an integer for the number of mistakes.

  ◊ Create a loop to repeat when all groups have not been found and the mistakes are less than four. Use the appropriate variables and the len() function.

  ◊ In the loop, call the pretty_print.displayGrid() function using the appropriate arguments. Call the helper.getUnconnectedWords() function and the interface.getGuessList() function. Call the interface.checkConnection() function and use its return value to update the list or mistakes variables appropriately.

  ◊ After the loop, use branching to check the mistakes. If the user made 4 mistakes, then they did not win. Call the pretty_print.displayGrid() will only one argument. Print the following message to the user and call the savePuzzle() function with only one argument.

  ```
  Better luck next time.
  ```

  ◊ If the user did not make 4 mistakes, then they won. Call the pretty_print.displayGrid() with two arguments to match the first two parameters. Print the following message to the user and call the savePuzzle() function with two arguments.

  ```
  Congratulations!
  ```

  ◊ This function will be called in the main() function in the main_*last_first*.py file.

9. Create a Python file entitled **main_*last_first*.py** where *last* is your last/family name and *first* is your first name. Use lowercase letters. Make sure to add the proper comment block at the top of the file. If your names (first & last) are not included in the filename and comments, you will not earn the points.
   - Import the **helper** file in order for the main() function in this file to be able to call functions in the helper.py file that you have defined.
   - Import the **interface** file in order for the main() function in this file to be able to call functions in the interface.py file that you have defined.
   - Define the **main()** function.

     Function: main
     Parameter: None
     Return value: None
     This function is the starting point of the program.
     - ◊ Display the following message to the user:

       ```
       Connections!
       Group words that share a common thread.
       ```
     - ◊ Call the interface.**displayRules()** function to read the text file and display the game rules to the user.
     - ◊ Call the helper.**createPuzzleList()** function to read the CSV file and create the list of dictionaries. If your CSV file is named "puzzles.csv", then you do not need to have an argument. Let the function use the default value for the filename. The createPuzzleList() function has a return value, so make sure to capture it in a variable. The return value is a list of dictionaries where each dictionary holds the information for one puzzle. If you are having a hard time with this function and want to test your other functions, you can call the pretty_print.loadData() function with no arguments, which will return the list of puzzle dictionaries. Calling loadData() is optional, but to do so you need to import pretty_print in this file and add the puzzles.bin file to your project_*last_first* directory.
     - ◊ Call the interface.**pickPuzzle()** function to pick a puzzle. Use the appropriate argument and capture the return value (a puzzle dictionary) in a variable.
     - ◊ Call the interface.**playGame()** function with the appropriate argument.
   - Call the **main()** function.

10. Be sure to comment your code. This means that there should be comments at the top of the files you created as well as throughout your code. Put a comment block at the top of each Python file. **Put a comment block before each function definition stating the function name, parameters, return values, and what that function does.** You can copy and paste text from these instructions. Points will be deducted for not having comments **before** the function definitions.
11. Follow coding conventions. You should use camelCase or snake_case for variable names. You are welcome to create any variables that you need. Variables should NOT start with an uppercase letter. **Function names must use camelCase.**
12. Test the program. Look at the Sample Output below. Projects that do not run are subject to 15% penalty.
13. Prepare your submission:
    - Find the **project_*last_first*** folder on your computer and compress it. This cannot be done within PyCharm. This folder MUST have the following files: **helper.py**, **interface.py**, **main_*last_first*.py**, **puzzles.csv**, **rules.txt**, and **pretty_print.py**. Your folder may contain the puzzles.bin file, but it is not required. If you are completing the extra credit, which is described in the last page(s) of this document, you should also have a file named extra_credit.py in the folder. Remove any other files or folders.
    - On Windows, use **File Explorer** to select the folder. Right-click and select the Send to -> Compressed (zipped) folder option. This will create a zip file.
    - On Mac OS, use **Finder** to select the folder. Right-click and select the Compress "*FolderName*" option. This will create a zip file.
14. Upload the zip file to your Brightspace section:
    - On Brightspace, click on the **Content** item.
    - Click on the **Final Project** item.
    - Click on the **Project** item.
    - Use the **Upload** button and select the zip file or drag the zip file into the dashed area.
    - Click the **Submit** button.

Grading

- This project is worth 100 points. With extra credit, you may earn up to 110 points.
- Make sure to submit your project correctly as described above. Points will be taken off for improper submission. Make sure your name and information are in the comments at the top of each Python file. If your name is not in the comments, you will receive 0 points.
- Any submission referencing past versions of this final project will receive 0 points and will be reported to the Office of Academic Integrity.
- Using concepts outside of the course materials will incur significant deductions and may result in being reported to the Office of Academic Integrity. It is impossible to list all the concepts you are not allowed to use. Do not use the following (this is not an exhaustive list): f-strings; the str.format() method; **`while True:`** loops; **`break`**, **`continue`**, **`pass`**, or **`with`** statements; is operator, tuples; sets; list or dict comprehension; the file.write() method; exceptions; the sorted() function; inline if; and the **__name__** conditional.
- You may NOT import and use the CSV module or other modules to process the CSV file.
- Do not use AI tools such as ChatGPT. Doing so is an academic integrity violation.

Category	Item	Points
helper.py	createPuzzleList()	12
	getPuzzle()	6
	getColor()	3
	getConnection()	3
	getWordsInGroup()	7
	getUnconnectedWords()	6
interface.py	displayRules()	6
	isValidNumber()	6
	pickPuzzle()	6
	getGuessList()	8
	checkConnection()	6
	savePuzzle()	6
	playGame()	12
main_*last_first*.py	main()	3
all py files	comments	6
	style (coding conventions)	4
	Total	100

Extra Credit

Add additional features for the user. Define and call functions to implement the features.

Add your new functions to a new file named extra_credit.py. Your name must be in the comment block at the top of the extra_credit.py file to earn points. Update your main_*last_first*.py file to import the file and call the new function(s) you define. Do not modify your helper.py and interface.py files. You can call functions in those files.

Since this is for extra credit, you will not be given detailed instructions on how to implement these. You should be able to implement these on your own. Instructors and learning assistants are not required to help you with extra credit. If you use concepts outside of the course materials in the final project, you are not eligible for earning extra credit. This content is protected and may not be shared, uploaded, or distributed. Posting this project on sites such as Chegg and Course Hero is an academic integrity violation. Do NOT use an AI tool such as ChatGPT.

Allow the user to add puzzles – 10 points

- Define a function (or more) to allow the user to enter in a puzzle with four groups of four by getting input from the user. Allow them to enter multiple puzzles. You can have your code write each new puzzle to a new file in the same format as puzzles.csv (use a different filename, start num at 1), append to the puzzles.csv file (num has to be greater than 508), create a list of dictionaries just like puzzleList, or add new puzzle dictionaries to puzzleList. If you write to a new file, you can call the createPuzzleList() function. If you append to the puzzles.csv file, call the createPuzzleList() function after.
- Update the main() function to use your function(s).

Sample Output for Extra Credit (user input shown in green)

```
Extra Credit - Add puzzles
Enter a number: 509
Enter a date: 2024 Nov 1

Enter connection1: PROGRESS SLOWLY
Enter word11: crawl
Enter word12: creep
Enter word13: drag
Enter word14: inch

Enter connection2: Ways to order a beer
Enter word21: bottle
Enter word22: can
Enter word23: draft
Enter word24: tap

Enter connection3: cheesy corn snack unit
Enter word31: BALL
Enter word32: CURL
Enter word33: Doodle
Enter word34: Puff

Enter connection4: ___ EFFECT
Enter word41: butterfly
Enter word42: domino
Enter word43: halo
Enter word44: placebo

Do you want to enter another puzzle? N

Enter a puzzle number (1-509): 509

               Create four groups of four!
CURL           CRAWL          PLACEBO        INCH
DOODLE         HALO           DRAFT          DRAG
BALL           DOMINO         BOTTLE         CREEP
PUFF           BUTTERFLY      CAN            TAP
               Mistakes remaining: * * * *
```