Exam Instructions

This exam consists of **three programming questions**. You have **2.5 hours** to complete and submit your answers.

- **Open-Book Policy**: You may consult course materials, such as the textbook, lecture slides, or online resources.
- **Restrictions**: The use of generative AI tools or code-generation software is strictly prohibited. Using such tools will be considered **cheating** and will result in a grade of zero.

Submission Guidelines:

1. Submit one Python (.py) or text (.txt) file per question:
    - Name each file as **q1**, **q2**, and **q3** respectively.
2. **File Integrity**:
    - Ensure that each file contains your complete solution.
    - Verify that all files have been saved properly and are not corrupted.
    - **Note**: Files that are empty or cannot be opened due to corruption or save issues will receive a grade of zero.

Please manage your time carefully and double-check all files before submission.

Question 1: Draw a Colorful Spiral Pattern – 10 Points

Create a Python program using **Turtle Graphics** to draw a colorful spiral pattern. The spiral should consist of multiple squares that gradually increase in size and change color as they rotate around the center.

Requirements:

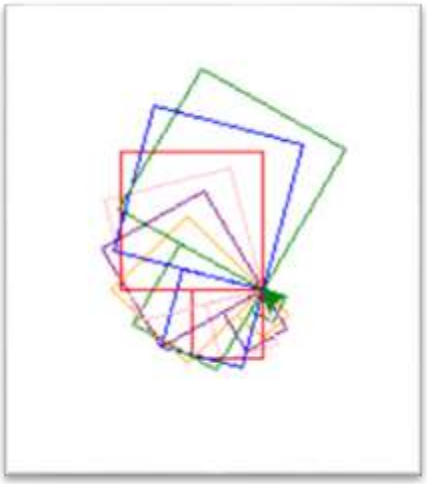
1. Define a function called `draw_square(turtle, side_length, color)` that:
    - Accepts a `turtle` object, a `side_length` for the square, and a `color`.
    - Sets the turtle color, and draws a square with the given side length.
2. Define a function called `draw_spiral(num_squares, size_increment)` that:
    - Accepts `num_squares` for the total number of squares to draw and a `size_increment` to increase the side length of each square in the spiral.
    - Calls `draw_square()` in a loop, increasing the size and rotating the turtle by a small angle after each square.
3. Use the `draw_spiral()` function with at least 15 squares and a size increment of 5 pixels per square.

Hints:

- Use `turtle.color()` to set the turtle's color. Experiment with colors by changing the RGB values or using random colors.
- Use `turtle.right(angle)` to rotate the turtle after each square to create a spiral effect.

Expected Outcome:

The program should draw a colorful spiral of squares, each one slightly larger and rotated. The design will look vibrant as the squares change colors in sequence. Here is an expected output.



Grading Rubric for Question 1 (10 Points total)

Criteria	Points	Description
Functionality	**4**	The program successfully draws a colorful spiral pattern with squares that increase in size and rotate.
		- 4 points: Fully functional spiral with correct colors, rotation, and size increase.
		- 3 points: Mostly functional with minor issues (e.g., slight inaccuracies in rotation or color sequence).
		- 2 points: Partially functional; pattern is incomplete or significantly inaccurate.
		- 1 point: Attempted but barely functional; very minimal output.
		- 0 points: No output or pattern produced.
Function Structure	**3**	The code is well-structured using functions as specified in the instructions.
		- 3 points: Both functions (`draw_square` and `draw_spiral`) are correctly defined and used appropriately.
		- 2 points: Functions are mostly used as specified but may be missing one part (e.g., not using color parameter).
		- 1 point: Only one function used, or functions do not adhere to the instructions.
Code Readability	**2**	The code is well-organized, with clear variable names and comments explaining each function's purpose.
		- 2 points: Clear, organized, and fully commented code.
		- 1 point: Mostly clear code but lacking comments or having unclear variable names.
		- 0 points: Hard to read or understand, lacking any comments or meaningful organization.
Creativity and Design	**1**	Shows creativity in the choice of colors, rotation angles, or overall design.
		- 1 point: Colors, rotation angles, or other elements enhance the visual appeal of the spiral.
		- 0 points: Very minimal or no effort toward creativity; the pattern appears plain or lacks variation.

Question 2: Tkinter Widget

Implement a Python GUI application using **Tkinter** with a widget called `RoundRobin`, which includes an **Entry** widget and a **Button** widget labeled `"Next"`.

The program should use a global, non-empty list of names as follows:

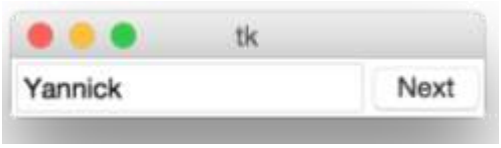
```
lst = ['Zoe', 'Yannick', 'Xena', 'Wendy', 'Vince']
```

Requirements:

1. **Function-Based Structure:**
   - Define a function named `create_round_robin(lst)` that accepts a list of items.
   - This function should:
     - Initialize the GUI window with an **Entry** widget and a **Button** labeled `"Next"`.
     - Display items from the list `lst` in the **Entry** widget in a round-robin fashion when the `"Next"` button is clicked.
2. **Display and Cycle Through List Items:**
   - When the program starts, the **Entry** widget should be empty.
   - Each time the `"Next"` button is clicked, the next item in the list `lst` should be displayed in the **Entry** widget.
   - After reaching the end of the list, clicking `"Next"` again should cycle back to the beginning, displaying the list items in **round-robin fashion**.
3. **Code Execution:**
   - Call the function to initialize and display the GUI widget by running:
   - The Entry widget should display nothing at startup:

   - When the “Next” button is clicked, the first item in the list should be displayed:

   - Clicking the “Next” button again should result in the next list item being displayed:



- In general, every button click should result in the next listitem being displayed, in round robin fashion (i.e., In case the entry displays the last item in the list, clicking the Next button one more time should display the first item in the list).

Question 2 Grading Rubric (10 Points Total)

Criteria	Pts	Description
Completeness	**3**	The program meets all requirements, including correct widget structure and functionality.
		- 3 points: All requirements are met, including initialization, display, and cycling behavior in round-robin order.
		- 2 points: Most requirements are met, but some minor functionality is missing or incomplete (e.g., initialization or list cycling doesn't fully work as expected).
		- 1 point: Program has significant missing components, such as GUI setup or button functionality.
Correctness	**3**	Program functions correctly and accurately displays items in round-robin order when the button is clicked.
		- 3 points: Each click reliably cycles through the list in order, looping back to the first item after the last.
		- 2 points: Mostly correct, but may skip items or not loop back as expected on the last item.
		- 1 point: Incorrect functionality; items are not displayed or cycled as required.
Function Design	**2**	The create_round_robin function is well-structured, and encapsulates the GUI setup and button functionality effectively.
		- 2 points: Function is correctly implemented, encapsulating all relevant functionality in create_round_robin.
		- 1 point: Function is present but lacks complete structure, e.g., not fully encapsulating the GUI setup or display logic.
Code Readability and Style	**1**	The code is well-organized, with clear, descriptive variable names and logical structure.
		- 1 point: Code is well-organized and readable, with clear variable names and logical organization.
		- 0 points: Code is poorly organized or difficult to understand.
Documentation	**1**	Code includes meaningful comments and a docstring explaining the purpose of create_round_robin.
		- 1 point: Clear docstring for the create_round_robin function and comments explaining key parts of the code.
		- 0 points: No comments or documentation provided.

Question 3: Text File Processing (10 points total)

Write a Python program that processes a text file to find the most frequent meaningful word. Specifically, your program should open a text file, read its content, and filter out non-functional terms such as `"a"`, `"the"`, `"an"`, `"in"`, `"of"`, `"to"`, `"and"`, and `"is"`. For the remaining words, the program should find and display the word with the highest frequency.

Requirements:

1. **Function Design**:
   - Define a function `most_frequent_word(filename, stop_words)` that:
     - Takes the name of the text file (`filename`) and a list of non-functional words (`stop_words`).
     - Opens and reads the file content, removing any punctuation and ignoring case.
     - Filters out any word found in the `stop_words` list.
     - Returns the word with the highest frequency from the remaining words, along with its frequency count.
2. **Ignore Case and Punctuation**:
   - Your program should treat words case-insensitively, so `"The"` and `"the"` should be considered the same.
   - Ignore any punctuation when counting word frequencies.
3. **Example Execution**:
   - Use the function to find the most frequent meaningful word in a sample file, and print the result in the format:

     ```
     The most frequent word is: 'word' with a frequency of X
     ```

4. **Sample Code**:

```
stop_words = ["a", "the", "an", "in", "of", "to", "and",
"is"]
filename = "sample.txt"
print(most_frequent_word(filename, stop_words))
```

Sample Output:

If the file contains the following text:

```
The quick brown fox jumps over the lazy dog. The dog was not
amused by the fox.
```

The output could be:

```
The most frequent word is: 'fox' with a frequency of 2
```

Question 3 Grading Rubric (10 points total)

Criteria	Points	Description
File Reading and Processing	**3**	The program correctly opens, reads, and processes the file content.
		- 3 points: Successfully reads and processes the file.
		- 2 points: Reads the file but may have minor issues (e.g., does not handle punctuation correctly).
		- 1 point: Attempts to read the file but contains significant issues.
Stop Words Filtering	**2**	The program successfully filters out the specified non-functional terms.
		- 2 points: Filters out all specified stop words correctly.
		- 1 point: Partially filters stop words, missing some or adding extra words.
Word Frequency Counting	**3**	The program correctly counts and identifies the most frequent word from the remaining terms.
		- 3 points: Accurate word frequency count and identifies the correct most frequent term.
		- 2 points: Counts most words correctly but may have minor issues with frequency calculation.
		- 1 point: Incorrect frequency count or fails to identify the correct most frequent term.
Function Design and Structure	**1**	The most_frequent_word function is well-structured and appropriately designed.
		- 1 point: Function is properly structured and encapsulates the logic effectively.
		- 0 points: Function is present but lacks clarity or structure.
Code Readability and Comments	**1**	Code is well-organized, with clear variable names and comments explaining each part of the process.
		- 1 point: Code is readable and includes comments.
		- 0 points: Code is difficult to read or lacks comments.

Good Luck!