Name: Youssef Salem Anwer Salem Hassan

General Requirements for All Tasks (if you didn't follow them you will be deducted):

- Use Python's typing module to annotate variables and function arguments and return types.
- Include docstrings (using triple-quoted strings) for all classes and public methods, explaining their purpose and usage.
- Include inline, multi-line comments where appropriate.
- Write clean, readable, and PEP 8-compliant code.

Task 1: Class and Inheritance

Description: Create two classes: `Shape` and `Rectangle`. The `Shape` class should be an **abstract** representation of a shape, and `Rectangle` should **inherit** from an abstract class **Shape** and implement a method to compute the area and perimeter of a rectangle.

Details:

- Shape Class:
  - `__init__` takes no parameters.
  - A method area() that raises NotImplementedError.
  - Include a docstring describing the class.
- Rectangle Class (inherits from Shape):
  - `__init__(self, width: float, height: float)`.
  - define the attributes of the Rectangle class as **private** inside `__init__` and you cannot set their values directly from the outside of the class, that means their names starts with `__`.
  - Implements `area()` which returns **self.width * self.height**.
  - Implments `perimeter()` which returns **2 * (self.length + self.width)**
  - Try to print the value of the width and height using `print(f"The width of the Rectangle is {rect.width} and height of the Rectangle is {rect.height}")` without getting any error.
  - Try to set the width and height using the defined object from the class like rect.width = 10 or rect.height = 20, without getting any error.
  - Include a docstring explaining the class and method.

```
from abc import ABC, abstractmethod

class Shape(ABC):
    """
    Abstract base class representing a geometric shape.
    """

    @abstractmethod
    def area(self):
        """
        Calculate the area of the shape.
        Must be implemented by subclasses.
        """
        raise NotImplementedError("Subclasses must implement this method")

class Rectangle(Shape):
    """
    A class representing a rectangle, inheriting from Shape.

    Attributes:
        __width (float): The width of the rectangle (private).
        __height (float): The height of the rectangle (private).
    """

    def __init__(self, width: float, height: float):
        """
        Initialize the rectangle with width and height.

        Args:
            width (float): The width of the rectangle.
            height (float): The height of the rectangle.
        """
        self.__width = width
        self.__height = height

    @property
    def width(self):
        """
        The width property getter.

        Returns:
            float: The width of the rectangle.
        """
        return self.__width

    @width.setter
    def width(self, value):
        """
        The width property setter.

        Args:
            value (float): The new width of the rectangle.
        """
        self.__width = value

    @property
    def height(self):
        """
        The height property getter.
```

```
        Returns:
            float: The height of the rectangle.
        """
        return self.__height

    @height.setter
    def height(self, value):
        """
        The height property setter.

        Args:
            value (float): The new height of the rectangle.
        """
        self.__height = value

    def area(self):
        """
        Calculate the area of the rectangle.

        Returns:
            float: The area of the rectangle.
        """
        return self.__width * self.__height

    def perimeter(self):
        """
        Calculate the perimeter of the rectangle.

        Returns:
            float: The perimeter of the rectangle.
        """
        return 2 * (self.__width + self.__height)

# Example Usage
rect = Rectangle(5, 10)
print(f"The width of the Rectangle is {rect.width} and height of the Rectangle is {rect.height}")
rect.width = 10
rect.height = 20
print(f"Updated width: {rect.width}, Updated height: {rect.height}")
print(f"Area: {rect.area()}")
print(f"Perimeter: {rect.perimeter()}")
```

```
The width of the Rectangle is 5 and height of the Rectangle is 10
Updated width: 10, Updated height: 20
Area: 200
Perimeter: 60
```

Task 2: Text File Operations

Description:

Create a class `TextFileStats` that manages a text file. It should handle file creation, appending content, deleting the file, and provide statistics about its contents.

Details:

- `__init__(self, file_path: str)` to store the file path.
- `create_file(self, initial_content: str = "") -> None:`
  - Creates the file at file_path. If initial_content is provided, write it into the file, and at the first line of the text file put the date and the time of the creation.
- `change_content(self, content: str = "") -> None:`
  - change the whole content of the file and overwrite to it content, and at the first line of the text file put the date and the time of the update.
- `append_content(self, content: str) -> None:`
  - Appends the given content to the file, and at the first line of the text file put the date and the time of the update.
- `delete_file(self) -> None:`
  - Deletes the file at the stored file_path.
- `line_count(self) -> int:`
  - Returns the number of lines in the file.
- `word_count(self) -> int:`
  - Returns the total number of words in the file (assuming words are separated by whitespace).
- Include docstrings for the class and all methods, and ensure that file operations are performed with with `open(...)` blocks and proper error handling where appropriate.

```
import os
from datetime import datetime

class TextFileStats:
    """
    A class to manage a text file, including operations like creation, appending,
    updating, deletion, and gathering statistics about the file's contents.
    """

    def __init__(self, file_path: str):
        """
        Initialize the TextFileStats with a file path.

        Args:
            file_path (str): The path to the text file.
```

```
        """
        self.file_path = file_path

    def create_file(self, initial_content: str = "") -> None:
        """
        Create a file at the specified path. If initial_content is provided,
        write it into the file, prefixed by the creation date and time.

        Args:
            initial_content (str): The initial content to write into the file.
        """
        try:
            with open(self.file_path, 'w') as file:
                timestamp = f"Created on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n"
                file.write(timestamp)
                file.write(initial_content)
        except Exception as e:
            print(f"Error creating file: {e}")

    def change_content(self, content: str = "") -> None:
        """
        Overwrite the file with new content, prefixed by the current date and time.

        Args:
            content (str): The new content to write into the file.
        """
        try:
            with open(self.file_path, 'w') as file:
                timestamp = f"Updated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n"
                file.write(timestamp)
                file.write(content)
        except Exception as e:
            print(f"Error changing file content: {e}")

    def append_content(self, content: str) -> None:
        """
        Append content to the file, prefixed by the current date and time.

        Args:
            content (str): The content to append to the file.
        """
        try:
            with open(self.file_path, 'a') as file:
                timestamp = f"Updated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n"
                file.write(timestamp)
                file.write(content)
        except Exception as e:
            print(f"Error appending to file: {e}")

    def delete_file(self) -> None:
        """
        Delete the file at the specified path.
        """
        try:
            if os.path.exists(self.file_path):
                os.remove(self.file_path)
            else:
                print("File does not exist.")
        except Exception as e:
            print(f"Error deleting file: {e}")

    def line_count(self) -> int:
        """
        Count the number of lines in the file.

        Returns:
            int: The number of lines in the file.
        """
        try:
            with open(self.file_path, 'r') as file:
                return sum(1 for _ in file)
        except Exception as e:
            print(f"Error counting lines: {e}")
            return 0

    def word_count(self) -> int:
        """
        Count the total number of words in the file.

        Returns:
            int: The total number of words in the file.
        """
        try:
            with open(self.file_path, 'r') as file:
                return sum(len(line.split()) for line in file)
        except Exception as e:
            print(f"Error counting words: {e}")
            return 0

# Example usage
file_manager = TextFileStats("example.txt")
file_manager.create_file("This is the initial content.\n")
file_manager.append_content("Here is some additional content.\n")
print(f"Line count: {file_manager.line_count()}")
print(f"Word count: {file_manager.word_count()}")
file_manager.change_content("This is a new content.")
print(f"Line count after update: {file_manager.line_count()}")
file_manager.delete_file()
```

```
Line count: 4
Word count: 18
Line count after update: 2
```

Task 3: Identify and Correct a Logical Error in a Sorting Algorithm

Description:

I Implemented a sorting function intended to sort a list of integers in ascending order. However, I got subtle logical error in the given code that prevents correct sorting, can you help me with:

- Identify the logical error by yourself.
- Correct the error and submit the fixed version.
- Include a docstring and appropriate comments and don't forget typing.

Faulty Sorting Function (Unaltered Version):

```
def sort(numbers):
    for i in range(1, len(numbers)):
        current = numbers[i]
        j = i - 1
        while j >= 1 and numbers[j] < current:
            numbers[j+1] = numbers[j]
            j -= 1
        numbers[j] = current
    return numbers

sort([50,54,24,87,59,78,14]) #output [78, 24, 24, 24, 24, 14, 14], correct output must be [14, 24, 50, 54, 59, 78, 87]
```

```
from typing import List

def sort(numbers: List[int]) -> List[int]:
    """
    Sort a list of integers in ascending order using an insertion sort algorithm.

    Args:
        numbers (List[int]): The list of integers to sort.

    Returns:
        List[int]: The sorted list of integers.
    """
    for i in range(1, len(numbers)):
        current = numbers[i]  # The current element to be placed in the correct position.
        j = i - 1
        # Ensure we compare all elements from the current position back to the start of the list.
        while j >= 0 and numbers[j] > current:
            numbers[j + 1] = numbers[j]  # Shift elements to make space for the current element.
            j -= 1
        # Place the current element in its correct position.
        numbers[j + 1] = current
    return numbers

# Example usage
if __name__ == "__main__":
    test_list = [50, 54, 24, 87, 59, 78, 14]
    print(f"Original list: {test_list}")
    sorted_list = sort(test_list)
    print(f"Sorted list: {sorted_list}")  # Expected output: [14, 24, 50, 54, 59, 78, 87]
```

```
Original list: [50, 54, 24, 87, 59, 78, 14]
Sorted list: [14, 24, 50, 54, 59, 78, 87]
```

Task 4: Identify and Correct a Logical Error in Another Algorithm

Description:

I Wrote a function that attempts to find the closest integer to a given target in a list. There is a subtle logical error in the comparison logic or the way values are updated. help me with:

- Identify the logical error by yourself.
- Correct the error and submit the fixed version.
- Include a docstring and appropriate comments and don't forget typing.

Faulty Function Example (Unaltered Version):

```
def find_closest(numbers, target):
    if not numbers:
        raise ValueError("The list 'numbers' cannot be empty.")

    closest = numbers[0]
    for num in numbers[1:]:
        if abs(num - target) < abs(closest - target) or (abs(num - target) == abs(closest - target) and num < closest):
            closest = closest
    return closest

    find_closest([1, 5, 9, 11, 10], 10) #output 1, but the closest number to 10 is 10 itself from the list.
    find_closest([1, 5, 7 ,9, 11, 10], 4) #output 1, but the closest number to 4 is 5.
    find_closest([1, 5, 7 ,9, 11, 10], 8) #output 1, but the closest number to 8 is 7 and 9, but we will chosse the smallest one.
```

```
from typing import List

def find_closest(numbers: List[int], target: int) -> int:
    """
    Find the closest integer to a given target in a list of numbers. If two numbers are equally close,
```

```
    the smaller number is returned.

    Args:
        numbers (List[int]): The list of integers to search.
        target (int): The target integer to find the closest number to.

    Returns:
        int: The closest integer to the target.

    Raises:
        ValueError: If the input list is empty.
    """
    if not numbers:
        raise ValueError("The list 'numbers' cannot be empty.")

    closest = numbers[0]
    for num in numbers[1:]:
        # Check if the current number is closer to the target.
        if abs(num - target) < abs(closest - target) or (abs(num - target) == abs(closest - target) and num < closest):
            closest = num  # Update closest to the current number if conditions are met.
    return closest

# Example usage
if __name__ == "__main__":
    print(find_closest([1, 5, 9, 11, 10], 10))  # Expected output: 10
    print(find_closest([1, 5, 7, 9, 11, 10], 4))  # Expected output: 5
    print(find_closest([1, 5, 7, 9, 11, 10], 8))  # Expected output: 7
```

```
10
5
7
```