Python Programs for practice

1. **Number Guessing Game:**
   - Write a program that prompts the user to guess a random number between 1 and 10 (inclusive).
   - Use a `while` loop to keep asking for guesses until the user enters the correct number.
   - Within the loop, use an `if` statement to check if the guess is correct, too high, or too low.
2. **Even or Odd Numbers:**
   - Write a program that takes a list of integers as input.
   - Use a `for` loop to iterate through the list.
   - Within the loop, use an `if` statement to check if each number is even or odd.
   - Print "Even" or "Odd" for each number.
3. **Grade Calculator:**
   - Write a program that prompts the user for their exam score.
   - Use an `if` statement with nested `elif` statements to assign letter grades based on the following criteria:
     1. A: 90-100
     2. B: 80-89
     3. C: 70-79
     4. D: 60-69
     5. F: Below 60
   - Print the corresponding letter grade.
4. **Positive, Negative, or Zero:**
   - Write a program that takes a list of numbers as input.
   - Use a `for` loop to iterate through the list.
   - Employ nested `if` statements to categorize each number as positive, negative, or zero.
   - Keep track of the counts for each category using variables.
   - Print the final counts of positive, negative, and zero numbers.
5. **Prime Number Checker:**
   - Write a program that prompts the user to enter a number.
   - Define a function `is_prime(n)` that takes a number `n` as input.
   - Within the function, use a `for` loop to iterate from 2 (exclusive) up to the square root of `n` (inclusive) using a loop control mechanism like `break`.
   - Inside the loop, use an `if` statement to check if `n` is divisible by any number between 2 and its square root.
   - If no divisors are found, return `True` from the function, indicating a prime number. Otherwise, return `False`.
   - Back in the main program, call the `is_prime(n)` function and print an appropriate message based on the result.
6. **Student Grade Statistics:**
   - Write a program that reads student data from a file (consider using `open()` and `readlines()`). Each line in the file should be formatted as "name score".
   - Use a `for` loop to iterate through each line of the file.
   - Split each line into `name` and `score` using string manipulation methods.
   - Apply the logic from question 3 to determine the letter grade for each student.
   - Calculate the average score for all students.

7. **Password Generator:**
   - Write a program that generates a secure password meeting specific criteria.
   - Requirements can include:
     1. Minimum length (e.g., 8 characters)
     2. At least one uppercase letter
     3. At least one lowercase letter
     4. At least one digit
     5. At least one special character (e.g., !@#$%^&*)
   - Use a `while` loop to keep generating passwords until a strong password is created.
   - Employ `random.choice()` to select random characters from different categories.
   - Check if the generated password meets all criteria using nested `if` statements and string methods.
   - Once a strong password is generated, print it.
8. **Text Analyzer:**
   - Write a program that analyzes a given text file (consider using `open()` and `read()`).
   - Use a `for` loop to iterate through each character in the text.
   - Keep track of the number of vowels, consonants, words, and sentences using counters.
   - Employ string methods to identify vowels, consonants, word boundaries (e.g., whitespace), and punctuation marks.
   - Print a report summarizing the analysis results (number of vowels, consonants, words, and sentences).
9. **List Creation and Access:**
   - Write a program that creates a list containing your favorite fruits.
   - Access and print the second element of the list.
   - Modify the last element of the list with a new fruit.
10. **List Iteration:**
    - Write a program that takes a list of numbers as input.
    - Use a `for` loop to iterate through the list and print each number doubled.
11. **List Concatenation:**
    - Create two lists: one with colors and another with numbers.
    - Use the `+` operator to combine these lists into a single list and print the result.
12. **List Slicing:**
    - Write a program that takes a list of words as input.
    - Extract a sublist containing only the second half of the words using slicing and print it.
13. **List Comprehensions:**
    - Create a list containing the squares of all numbers from 1 to 10 (inclusive) using list comprehension and print it.
    - Modify the previous program to create a new list containing only even numbers from 1 to 10 (inclusive) using list comprehension.
14. **List Membership and Removal:**
    - Write a program that takes a list of names and a specific name as input.
    - Use the `in` operator to check if the specific name exists in the list and print a message accordingly.
    - If the name exists, remove it from the list using the `remove()` method and print the updated list.

15. **List Sorting:**
    - Write a program that takes a list of mixed data types (e.g., strings, numbers) as input.
    - Use the `sorted()` function to sort the list in ascending order and print the result.
    - Modify the program to sort the list in descending order using an appropriate argument with `sorted()`.
16. **List Functions:**
    - Write a program that takes a list of numbers as input.
    - Use functions like `max()` and `min()` to find the largest and smallest numbers in the list and print them.
    - Calculate the sum and average of the numbers in the list using functions like `sum()` and division and print the results.
17. **List Manipulation with User Input:**

    Write a program that allows the user to enter a list of items (e.g., groceries) one by one until they choose to stop.Store the entered items in a list.

18. After the user finishes entering items, display the list and provide options to:
    - Add a new item to the list
    - Remove an item from the list
    - Print the list in a specific order (e.g., alphabetical)
19. **Creating and Writing to a File:**
    - Write a program that prompts the user for a message and creates a new text file named "message.txt".
    - Open the file in write mode ("w") and use the `write()` method to write the user's message to the file.
    - Close the file.
20. **Reading from a File:**
    - Write a program that opens an existing text file named "data.txt" in read mode ("r").
    - Read the entire content of the file using the `read()` method and store it in a variable.
    - Print the contents of the variable to display the file's data.
21. **Appending to a File:**
    - Write a program that opens an existing text file named "log.txt" in append mode ("a").
    - Prompt the user for a new log entry and write it to the file using the `write()` method.
    - Close the file.
22. **Line-by-Line Reading:**
    - Write a program that opens a text file named "numbers.txt" containing numbers (one per line).
    - Use a `for` loop with the `readlines()` method to iterate through each line in the file.
    - Convert each line from a string to an integer using `int()` and calculate the sum of all the numbers.
    - Print the final sum.
23. **File with Delimiters:**

- Write a program that reads a CSV file named "data.csv" containing comma-separated values (e.g., name, age, city).
    - Use the `csv` module to open the file and iterate through each row using a `for` loop.
    - Access each column's value within a row and print them in a formatted way (e.g., "Name: {name}, Age: {age}, City: {city}").

24. **Copying a File:**
    - Write a program that takes two file names as input: one for the source file and another for the destination.
    - Open the source file in read mode ("r") and the destination file in write mode ("w").
    - Use a `for` loop with the `read()` method in chunks (e.g., 1024 bytes) to read from the source file and write the content to the destination file.
    - Close both files.
25. **Load and Print DataFrame:**
    - Import the `pandas` library as `pd`.
    - Write code to read a CSV file named "data.csv" into a Pandas DataFrame named `df`.
    - Print the first few rows of the DataFrame using `df.head()`.
26. **Accessing Data by Column Name:**
    - Load a CSV file containing student data (name, age, grade).
    - Access and print a specific column, like the "grade" column, using `df["grade"]`.
27. **Descriptive Statistics:**
    - Load a CSV file containing sales data (product, quantity sold, price).
    - Calculate summary statistics like mean, standard deviation, minimum, and maximum for the "quantity sold" column using `df["quantity sold"].describe()`.
28. **Basic Animal Inheritance:**
    - Create a base class `Animal` with attributes like `name` and `sound` (optional).
    - Define methods like `make_sound()` (optional) that can be overridden in subclasses.
    - Create subclasses `Dog` and `Cat` that inherit from `Animal`, potentially overriding `make_sound()` to print specific animal sounds.
    - Instantiate objects of the `Dog` and `Cat` classes and demonstrate polymorphism by calling `make_sound()` on both objects (even if it's not implemented in the base class).
29. **Shape Inheritance:**
    - Create a base class `Shape` with methods like `get_area()` that can be implemented by subclasses.
    - Create subclasses `Square` and `Circle` that inherit from `Shape` and implement `get_area()` based on their specific formulas.
    - Instantiate objects of `Square` and `Circle` and demonstrate polymorphism by calling `get_area()` on both objects, showcasing the different area calculations.