Question Bank on Module 3(Strings and File Handling)

1. **Discuss various methods in a string with an example program using at least 5 methods.**
   In Python, a string is a sequence of characters, which can be accessed, modified, and manipulated using various built-in methods. Here are some common string methods in Python with examples:
   a. **upper() and lower():** These methods return a new string with all letters in upper or lower case, respectively.

```
string = "Hello, World!"
print(string.upper())  # prints "HELLO, WORLD!"
print(string.lower())  # prints "hello, world!"
```

   b. **strip(), lstrip(), and rstrip():** These methods remove whitespace characters from the beginning and/or end of a string.

```
string = "   Hello, World!   "
print(string.strip())    # prints "Hello, World!"
print(string.lstrip())   # prints "Hello, World!   "
print(string.rstrip())   # prints "   Hello, World!"
```

   c. **replace():** This method returns a new string with all occurrences of a substring replaced by another substring.

```
string = "Hello, World!"
new_string = string.replace("World", "Python")
print(new_string)  # prints "Hello, Python!"
```

   d. **split() and join():** These methods are used to split a string into a list of substrings or to join a list of substrings into a single string.

```
string = "Hello, World!"
words = string.split(", ")
print(words)  # prints ["Hello", "World!"]

new_string = "-".join(words)
print(new_string)  # prints "Hello-World!"
```

   e. find(), index(), and count(): These methods are used to search for a substring within a string, and to count the occurrences of a substring within a string.

```
string = "Hello, World!"
index = string.find("World")
print(index)  # prints 7

index = string.index("World")
print(index)  # prints 7

count = string.count("l")
print(count)  # prints 3
```

Overall Example program using all the above methods:

```
string = "   The quick brown fox jumps over the lazy dog   "

# upper and lower
print(string.upper())  # prints "   THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG   "
print(string.lower())  # prints "   the quick brown fox jumps over the lazy dog   "

# strip, lstrip, and rstrip
print(string.strip())   # prints "The quick brown fox jumps over the lazy dog"
print(string.lstrip())  # prints "The quick brown fox jumps over the lazy dog   "
print(string.rstrip())  # prints "   The quick brown fox jumps over the lazy dog"

# replace
new_string = string.replace("lazy", "sleepy")
print(new_string)  # prints "   The quick brown fox jumps over the sleepy dog   "

# split and join
words = string.split()
print(words)  # prints ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy",
"dog"]

new_string = "-".join(words)
print(new_string)  # prints "The-quick-brown-fox-jumps-over-the-lazy-dog"

# find, index, and count
index = string.find("fox")
print(index)  # prints 18

index = string.index("fox")
print(index)  # prints 18

count = string.count("o")
print(count)  # prints 5
```

2. Relate String and List in Python

String:

String is a sequence of characters and it is represented within double quotes or single quotes. Strings are immutable.

Example: s=”hello”

List:

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type and it is mutable.

Example:
b= [’a’, ’b’, ’c’, ’d’, 1, 3]

In Python, strings and lists are both sequence types and share some similarities in their behaviors and methods. Here are some ways in which strings and lists are related:

Both are sequences: A string is a sequence of characters, while a list is a sequence of any type of values.

Indexing: Both strings and lists can be indexed. That means you can access individual elements of a string or a list using an index. For example, string[0] and list[0] both return the first element of the sequence.

Slicing: Both strings and lists can be sliced. That means you can create a new string or list by extracting a portion of the original sequence. For example, string[1:4] and list[1:4] both return a new sequence consisting of the second, third, and fourth elements of the original sequence.

Concatenation: Both strings and lists can be concatenated. That means you can create a new sequence by combining two or more sequences. For example, "Hello" + "World" returns the string "HelloWorld", and [1, 2] + [3, 4] returns the list [1, 2, 3, 4].

Iteration: Both strings and lists can be iterated over using a for loop. That means you can loop over all the elements in a string or a list.

Some methods: Some methods such as split() and join() can be used both with strings and lists. For example, string.split() and list.split() both return a list of substrings, and " ".join(list) and " ".join(string) both join a list or a string with spaces.

While strings and lists share some similarities, they are also different in some important ways. For example, strings are immutable, meaning that you cannot change the characters in a string, while lists are mutable, meaning that you can add, remove, or modify elements in a list. Additionally, strings have some methods that are specific to strings, such as upper() and lower(), while lists have some methods that are specific to lists, such as append() and sort().

3. Write a Python program to count the number of vowels in a string provided by the user

```
string = input("Enter a string: ")
vowels = "aeiouAEIOU"
count = 0

for char in string:
    if char in vowels:
        count += 1
```

```
print("Number of vowels in the string: ", count)
```

4. **Write a program that takes a sentence as input from the user and computes the frequency of each letter. Use a variable of dictionary type to maintain the count**

```
sentence = input("Enter a sentence: ")
letter_count = {}

for letter in sentence:
    if letter.isalpha():
        letter = letter.lower()
        if letter in letter_count:
            letter_count[letter] += 1
        else:
            letter_count[letter] = 1

print("Letter frequency:")
for letter, count in letter_count.items():
    print(letter, count)
```

5. **Analyze string slicing. Illustrate how it is done in python with an example**

In Python, strings are sequences of characters, which can be accessed, modified, and manipulated using a variety of built-in methods. A string can be sliced using the square bracket notation.

The syntax is ***string[start:end:step]***, where ***start*** is the index of the first character to include in the slice (inclusive), ***end*** is the index of the last character to include in the slice (exclusive), and ***step*** is the increment between characters. For example:

```
string = "Hello, World!"
print(string[7:])   # prints "World!"
print(string[::2])  # prints "Hlo ol!"
```

6. **Explain the syntax of reading and writing files in python.**

Syntax for reading the contents of the file:

```
filehandle = open(filename, “r”)
data = filehandle.read()
```

where filename if the fully qualified name of the file & filehandle is the reference variable to the file object.

Example:

```
fp = open(“sample.txt”, “r”)
data = fp.read()
```

Syntax for writing the contents to the file:

```
filehandle = open(filename, “w”)
filehandle.write(data)
```

where filename if the fully qualified name of the file & filehandle is the reference variable to the file object.

Example:

```
fp  = open(“sample.txt”, “w+”)
fp.write(“Sample data to be written to the file”)
```

7. Describe the different access modes of the files with examples.

- **"r" (read-only):** This mode allows you to open a file for reading only. You cannot modify the contents of the file using this mode. Example: **file = open("example.txt", "r")**.
- **"w" (write-only):** This mode allows you to open a file for writing only. If the file already exists, its contents will be deleted before writing to it. If the file does not exist, it will be created. Example: **file = open("example.txt", "w")**.
- **"a" (append):** This mode allows you to open a file for appending new data to it. If the file already exists, new data will be added to the end of the file. If the file does not exist, it will be created. Example: **file = open("example.txt", "a")**.
- **"x" (exclusive creation):** This mode allows you to open a file for writing, but only if it does not already exist. If the file already exists, a **FileExistsError** exception will be raised. Example: **file = open("example.txt", "x")**.
- **"b" (binary mode):** This mode allows you to open a file in binary mode, which is used to read or write non-text files, such as images or audio. This mode can be used with any of the above modes, by adding "b" to the end of the mode string. Example: **file = open("example.jpg", "rb")**.
- **"+" (read-write mode):** This mode allows you to open a file for both reading and writing. This mode can be used with any of the above modes, by adding "+" to the end of the mode string. Example: **file = open("example.txt", "r+")**.

Example:

```
# Open a file in read-only mode and print its contents
file = open("example.txt", "r")
print(file.read())
file.close()

# Open a file in write-only mode and write some data to it
file = open("example.txt", "w")
file.write("This is some data.")
file.close()

# Open a file in append mode and add more data to it
file = open("example.txt", "a")
file.write("\nThis is more data.")
file.close()

# Open a file in exclusive creation mode and try to write to it
try:
   file = open("example.txt", "x")
   file.write("This will not work.")
   file.close()
except FileExistsError:
   print("File already exists.")

# Open a binary file in read mode and print its contents
file = open("example.jpg", "rb")
print(file.read())
file.close()
```

```
# Open a file in read-write mode and modify its contents
file = open("example.txt", "r+")
data = file.read()
data = data.replace("data", "text")
file.seek(0)
file.write(data)
file.close()
```

8. Discuss the following methods associated with the file object with suitable examples.
a) read() b) readline() c) readlines() d) seek() e) write()

a) **read():** The read() method reads the entire contents of a file and returns them as a string. Here's an example:

```
# Open a file and read its contents
with open("example.txt", "r") as file:
    data = file.read()
    print(data)
```

b) **readline():** The readline() method reads a single line from a file and returns it as a string. If you call this method again, it will return the next line in the file.

```
# Open a file and read its contents line by line
with open("example.txt", "r") as file:
    line = file.readline()
    while line:
        print(line)
        line = file.readline()
```

c) **readlines():** The readlines() method reads all the lines of a file and returns them as a list of strings.

```
# Open a file and read its contents into a list
with open("example.txt", "r") as file:
    lines = file.readlines()
    print(lines)
```

d) **seek():** The seek() method moves the file pointer to a specific position in the file. The file pointer is the position in the file where the next read or write operation will occur.

```
# Open a file and move the file pointer to a specific position
with open("example.txt", "r") as file:
    file.seek(5)
    data = file.read()
    print(data)
```

e) **write():** The write() method writes a string to a file. If the file does not exist, it will be created. If the file already exists, the contents will be overwritten.

```
# Open a file and write some data to it
with open("example.txt", "w") as file:
    file.write("This is some data.")
```

9. Write Python Program to reverse each word in "words.txt" file.

```
# Open the input and output files
with open("words.txt", "r") as input_file:
# Read each line from the input file
   for line in input_file:
      # Split the line into a list of words
      words = line.strip().split()
      # Reverse each word and write to the output file
      for word in words:
         reversed_words = word[::-1]
         print(reversed_words)
```

10. Write Python Program to count the Occurrences of each word and also count the number of words in a "quotes.txt" File.

```
# Open the input file
with open("quotes.txt", "r") as file:
   # Initialize a dictionary to store the word counts
   word_counts = {}

   # Loop through each line in the file
   for line in file:
      # Split the line into a list of words
      words = line.strip().split()

      # Loop through each word and update the word counts
      for word in words:
         if word not in word_counts:
            word_counts[word] = 1
         else:
            word_counts[word] += 1

   # Count the total number of words
   total_words = sum(word_counts.values())
   # Print the word counts and total number of words
   print("Word counts:")
   for word, count in word_counts.items():
      print(f"{word}: {count}")
   print(f"Total words: {total_words}")
```

11. Write Python Program to find the longest word in the specified file.

```
# Function to find the longest word in a file
def find_longest_word(filename):
   # Open the file in read mode
```

```
    with open(filename, "r") as file:
        # Initialize the longest word to an empty string
        longest_word = ""

        # Loop through each line in the file
        for line in file:
            # Split the line into a list of words
            words = line.strip().split()

            # Loop through each word and update the longest word
            for word in words:
                if len(word) > len(longest_word):
                    longest_word = word

    # Return the longest word
    return longest_word

# Example usage: find the longest word in "words.txt"
filename = "words.txt"
longest_word = find_longest_word(filename)
print(f"The longest word in {filename} is '{longest_word}' with {len(longest_word)} characters.")
```

12. Write a python program that counts the number of words in a file.

```
f=open("test.txt","r")
content =f.readline(20)
words =content.split()
print(words)
```

13. Write a python program to count number of lines, words and characters in a text file

```
# Function to count the number of lines, words, and characters in a file
def count_file_stats(filename):
    # Open the file in read mode
    with open(filename, "r") as file:
        # Initialize the line, word, and character counts to 0
        num_lines = 0
        num_words = 0
        num_chars = 0

        # Loop through each line in the file
        for line in file:
            # Increment the line count
            num_lines += 1

            # Split the line into a list of words
            words = line.strip().split()
```

```
        # Increment the word and character counts
        num_words += len(words)
        num_chars += len(line)

    # Return a dictionary of the counts
    return {"lines": num_lines, "words": num_words, "characters": num_chars}

# Example usage: count the lines, words, and characters in "text.txt"
filename = "text.txt"
file_stats = count_file_stats(filename)
print(f"{filename} contains {file_stats['lines']} lines, {file_stats['words']} words, and
{file_stats['characters']} characters.")
```

14. Write a python program to check whether the given string is palindrome or not

```
def is_palindrome(string):
    # Convert the string to lowercase and remove whitespace and punctuation
    string = string.lower().replace(" ", "").replace(",", "").replace(".", "")

    # Check if the string is equal to its reverse
    return string == string[::-1]

# Example usage: check if a string is a palindrome
string = input("Enter a String")
if is_palindrome(string):
    print(f"{string} is a palindrome.")
else:
    print(f"{string} is not a palindrome.")
```