v

Author : DSALGO

PYTHON UNIT 4

File handling

- File handling in Python refers to the manipulation and management of files,
- such as reading from or writing to them.
- Python provides built-in functions and methods to perform various file operations.

1. opening file
2. Reading file
3. Writing file
4. file navigation
5. checking file existance

- **Advantages**
  1. Versatility 2. ease to use 3. portability 4. Flexibility 5.integration
- **Disadvantages**
  1. performance 2.error handling 3. security Risks 4. Limited File Locking 5. Resource management

Opening Files

- You can use the open() function to open a file.
- It takes two parameters:
- file path 2. mode[r,w, a]
- **syntax**:

```
file = open('filename.txt', 'r')
```

Closing Files

- After using read(), it's important to close the file using the close() method to release system resources
- **syntax**:

```
file.close()
```

read() :

- read() function in Python is used to read data from a file.
- read() function is called on a file object and takes an optional parameter specifying the number of bytes to read.
- If no size is specified, it reads the entire file.
- **Syntax**:

```
file_object.read(size)
```

- **Example 1**:

```
file = open('example.txt', 'r')
data = file.read() # Reads the entire contents
file.close()
```

- **Example 2: (file cursor)**

```
file = open('example.txt', 'r')
data1 = file.read(10)  # Reads the first 10 bytes
data2 = file.read(10)  # Reads the next 10 bytes
file.close()
```

read() with splitlines():

- You can also use read() along with splitlines() to read lines from the file
- **Example:**

```
file = open('example.txt', 'r')
lines = file.read().splitlines()  # Reads lines from the file into a list
file.close()
```

readline():

- Reads a single line from the file.
- Moves the file cursor to the beginning of the next line after reading.
- Returns the line as a string, including the newline character ('\n'), unless the end of the file is reached, in which case an empty string is returned.
- **Example:**

```
file = open('example.txt', 'r')
lines = file.readline()#read first line of the file
lines = file.readline()#read second line of the file
```

readlines():

- Reads all lines from the file and returns them as a list of strings.
- Each string in the list represents a single line from the file, including the newline character ('\n').
- Useful when you want to iterate over lines in a file or process them individually.
- Example:

```
file = open('example.txt', 'r')
lines = file.readlines()  # Reads all lines into a list
file.close()
```

Difference b/w read(), readline() , readlines() function:

- **readline()** when you want to read lines one by one.
- **read()** when you want to read a specific number of bytes or the entire file.
- **readlines()** when you want to read all lines into a list for further processing.

write()

- Writes a string to the file.
- If the file is opened in text mode ('t'), you can only write strings to it.
- If the file is opened in binary mode ('b'), you can write bytes-like objects (e.g**., bytes or bytearray**).
- **Examples:**

```
file = open('example.txt', 'w')  # Open file in write mode
file.write("Hello, world!")  # Write a string to the file
file.close()
```

writelines():

- Writes a list of strings to the file.
- Each string in the list represents a line, and newline characters ('\n') must be included if needed.
- **Example:**

```
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
file = open('example.txt', 'w')
file.writelines(lines)  # Write multiple lines to the file
file.close()
```

Write using print()

- You can use print() function with file parameter to write to file.
- By default, it adds a newline character ('\n') after each print unless specified otherwise
- **Example:**

```
file = open('example.txt', 'w')
print("Hello, world!", file=file)  # Write to file using print function
file.close()
```

Appending to a File

- If you want to append data to an existing file without overwriting its contents, open it in append mode ('a') instead of write mode ('w').
- **Example:**

```
file = open('example.txt', 'a')  # Open file in append mode
file.write("Appending new content!")  # Append to the file
file.close()
```

Write and read using *with* Statement:

- You can use the with statement to automatically close the file after writing, ensuring proper resource management
- **Examples:**

```
with open('example.txt', 'w') as file:
    file.write("Using 'with' statement for file writing.")

with open('filename.txt', 'r') as file:
    content = file.read()
    # File automatically closed after this block
```

Seek() function:

- Manipulating the file pointer using the seek() function in Python allows you to move the cursor position within the file
- **Examples:**

```
# Open a file in read mode
with open('example.txt', 'r') as file:
    # Read the first 10 bytes
    data1 = file.read(10)

    # Get the current cursor position
    position = file.tell()
    print("Current position:", position)

    # Move the cursor to the beginning of the file
    file.seek(0)
    # Read the next 20 bytes from the beginning of the file
    data2 = file.read(20)

    # Print the read data
    print("Data 1:", data1)
    print("Data 2:", data2)
```

Description :

- We open a file named **example.txt** in read mode.
- We use read(10) to read the first 10 bytes from the file.
- We use tell() to get the current cursor position.
- We use seek(0) to move the cursor back to the beginning of the file.
- We use read(20) to read the next 20 bytes from the beginning of the file.
- Finally, we print the read data.

Q. Create a Python program that reads the content of a source file and writes it to a destination file.

```
def copy_file(source_file, destination_file):
    try:
        with open(source_file, 'r') as source:
            with open(destination_file, 'w') as destination:
                for line in source:
                    destination.write(line)
        print(f"Content copied from '{source_file}' to '{destination_file}' successfully.")
    except FileNotFoundError:
        print("One of the files could not be found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage:
source_file = input("Enter the source file name: ")
destination_file = input("Enter the destination file name: ")

#function call
copy_file(source_file, destination_file)
```

Q.Develop a Python program that reads a text file and prints the longest line along with its length

```
def longest_line(file_name):
    try:
        with open(file_name, 'r') as file:
            lines = file.readlines()
            longest_length = 0
            longest_line = ""

            for line in lines:
                line_length = len(line.strip())
                if line_length > longest_length:
                    longest_length = line_length
                    longest_line = line.strip()

        print(f"The longest line is: '{longest_line}'")
        print(f"Its length is: {longest_length} characters")
    except FileNotFoundError:
        print(f"File '{file_name}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Q.Write a Python program to search for a specific string in a text file and print out the line(s) where it occurs.

```
# Example usage:
file_name = input("Enter the file name: ")
longest_line(file_name)
def search_string_in_file(file_name, search_string):
    try:
        with open(file_name, 'r') as file:
            for line_number, line in enumerate(file, 1):
                if search_string in line:
                    print(f"Found '{search_string}' in line {line_number}: {line.strip()}")
    except FileNotFoundError:
        print(f"File '{file_name}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage:
file_name = input("Enter the file name: ")
search_string = input("Enter the string to search: ")

search_string_in_file(file_name, search_string)
```

Q.Write a Python program to read a text file and count the total number of words in it.

Q.Design a Python program that encrypts the contents of a file using a simple substitution cipher and then decrypts it back to its original form.