1. Add the current date to the text file today.txt as a string.

Ans : We can use a programming language like Python to add the current date to the text file. Here's an example using Python:

from datetime import date # Get the current date current\_date = date.today() # Convert the date to a string date\_string = current\_date.strftime("%Y-%m-%d") # Open the file in append mode and write the date string with open("today.txt", "a") as file: file.write(date\_string + "\n")

In this example, the current date is obtained using the **date.today()** method. It is then converted to a string format using the **strftime** method with the specified format ("%Y-%m-%d" represents the year, month, and day). The file "today.txt" is then opened in append mode, and the date string is written to the file.

You can run this Python code on your local machine, ensuring that the file "today.txt" exists in the same directory as the script, and it will add the current date to the file.

2. Read the text file today.txt into the string today\_string

Ans : in Python that reads the contents of the "today.txt" file into a string variable called today\_string:

# Read the contents of the file into a string with open("today.txt", "r") as file: today\_string = file.read() # Print the contents of the string print(today\_string)

In this code snippet, the file "today.txt" is opened in read mode using the **open()** function. The **read()** method is then called on the file object to read the entire contents of the file into the **today\_string** variable. Finally, you can print the **today\_string** variable or perform any other operations you want with its contents.

Make sure the "today.txt" file exists in the same directory as the Python script or provide the full file path if it's located elsewhere.

3. Parse the date from today\_string.

Ans : To parse the date from the today\_string, you can use the datetime module in Python. Here's an example of how you can do it:

from datetime import datetime # Assuming the date format in today\_string is "%Y-%m-%d" date\_format = "%Y-%m-%d" # Parse the date from today\_string parsed\_date = datetime.strptime(today\_string.strip(), date\_format).date() # Print the parsed date print(parsed\_date)

In this code snippet, we assume that the date format in **today\_string** is **"%Y-%m-%d"**, which corresponds to the format used in the previous example to write the date to the file.

The **strptime()** method is used to parse the date from the **today\_string**. The **strip()** function is called to remove any leading or trailing whitespace from the string before parsing. The parsed date is stored in the **parsed\_date** variable as a **date** object.

You can then use or manipulate the **parsed\_date** variable as needed. In the example above, it is printed using the **print()** function.

4. List the files in your current directory

Ans : Here's an example using the os module:

import os # Get the current directory current\_directory = os.getcwd() # List files in the current directory file\_list = os.listdir(current\_directory) # Print the file names for file\_name in file\_list: print(file\_name)

In this example, the **os** module is imported to work with the operating system. The **getcwd()** function is used to get the current working directory. Then, the **listdir()** function is called with the current directory as an argument to obtain a list of files in that directory.

Finally, a loop is used to iterate over the **file\_list** and print each file name. You can modify this example to suit your specific needs, such as filtering the files based on certain criteria or performing additional operations on the file names.

5. Create a list of all of the files in your parent directory (minimum five files should be available).

Ans : We can modify the code snippet according to your specific scenario.

import os # Get the parent directory parent\_directory = os.path.dirname(os.getcwd()) # List files in the parent directory file\_list = os.listdir(parent\_directory) # Print the file names for file\_name in file\_list: print(file\_name)

In this example, the **os** module is used to interact with the operating system. The **dirname()** function is called on the current working directory obtained from **os.getcwd()** to get the parent directory path. Then, the **listdir()** function is used with the parent directory path to retrieve a list of files in that directory.

Finally, a loop is used to iterate over the **file\_list** and print each file name. You can modify this code to perform different operations or filter the files based on your specific requirements.

6. Use multiprocessing to create three separate processes. Make each one wait a random number of seconds between one and five, print the current time, and then exit.

Ans : Here's an example using the multiprocessing module in Python to create three separate processes. Each process will wait for a random number of seconds between one and five, print the current time, and then exit:

import multiprocessing import random import time from datetime import datetime # Function to print current time and exit def print\_current\_time(): wait\_time = random.randint(1, 5) # Generate random wait time between 1 and 5 seconds time.sleep(wait\_time) # Wait for the specified time current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Get current time print(f"Process ID: {multiprocessing.current\_process().pid}, Current Time: {current\_time}") # Print process ID and current time # Create three separate processes processes = [] for \_ in range(3): process = multiprocessing.Process(target=print\_current\_time) processes.append(process) process.start() # Wait for all processes to finish for process in processes: process.join()

In this code snippet, the **print\_current\_time** function is defined, which generates a random wait time between 1 and 5 seconds using **random.randint()** and sleeps for that duration using **time.sleep()**. Then, it retrieves the current time using **datetime.now()** and prints the process ID and the current time.

Next, three separate processes are created using **multiprocessing.Process**. The **target** parameter is set to **print\_current\_time**, indicating that each process will execute that function. The processes are started using **process.start()**.

Finally, the main process waits for all the child processes to finish using **process.join()**.

When you run this code, you will see three separate processes printing their process ID and the current time after waiting for a random duration between 1 and 5 seconds.

7. Create a date object of your day of birth.

Ans : an example of creating a date object for a specific date. Let's assume the date of birth is September 1, 2000. Here's how you can create a date object using Python:

from datetime import date # Create a date object for the day of birth birth\_date = date(2000, 9, 1) # Print the birth date print(birth\_date)

In this example, the **date** module is imported from the **datetime** package. The **date()** constructor is then used to create a **date** object with the year, month, and day specified as arguments. In this case, the date of birth is set to September 1, 2000.

Finally, the **birth\_date** object is printed using the **print()** function.

8. What day of the week was your day of birth?

Ans : an example of how to determine the day of the week for a given date in Python. Let's assume the date of birth is September 1, 2000. Here's how you can find the day of the week:

from datetime import date # Create a date object for the day of birth birth\_date = date(2000, 9, 1) # Get the day of the week (0 = Monday, 1 = Tuesday, ..., 6 = Sunday) day\_of\_week = birth\_date.weekday() # Define a list of weekdays weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'] # Get the weekday name weekday\_name = weekdays[day\_of\_week] # Print the day of the week print("The day of the week for my birthdate was:", weekday\_name)

In this example, the **date** module is imported from the **datetime** package. The **date()** constructor is used to create a **date** object representing the date of birth.

The **weekday()** method is called on the **birth\_date** object, which returns the day of the week as an integer (where 0 represents Monday and 6 represents Sunday).

To map the integer value to the corresponding weekday name, a list of weekdays is defined. The weekday name is retrieved from the **weekdays** list based on the **day\_of\_week** value.

Finally, the day of the week is printed using the **print()** function.

9. When will you be (or when were you) 10,000 days old?

Ans : To calculate the date that is 10,000 days after September 2, 1998, you can use Python's datetime module. Here's an example:

from datetime import datetime, timedelta # Starting date start\_date = datetime(1998, 9, 2) # Calculate the date that is 10,000 days after the starting date result\_date = start\_date + timedelta(days=10000) # Format the result date as a string result\_date\_string = result\_date.strftime("%m-%d-%Y") # Print the result print("The date that is 10,000 days after September 2, 1998, is:", result\_date\_string)

In this example, the **datetime** module is imported, and the starting date is set as September 2, 1998, using the **datetime()** constructor.

To calculate the date that is 10,000 days after the starting date, we add a **timedelta** of 10,000 days to the starting date using the **timedelta()** function.

The result date is then formatted as a string using the **strftime()** method, specifying the desired date format ("%m-%d-%Y" in this case).

Finally, the result is printed, displaying the date that is 10,000 days after September 2, 1998.