**Working with some libraries**

**Task:**

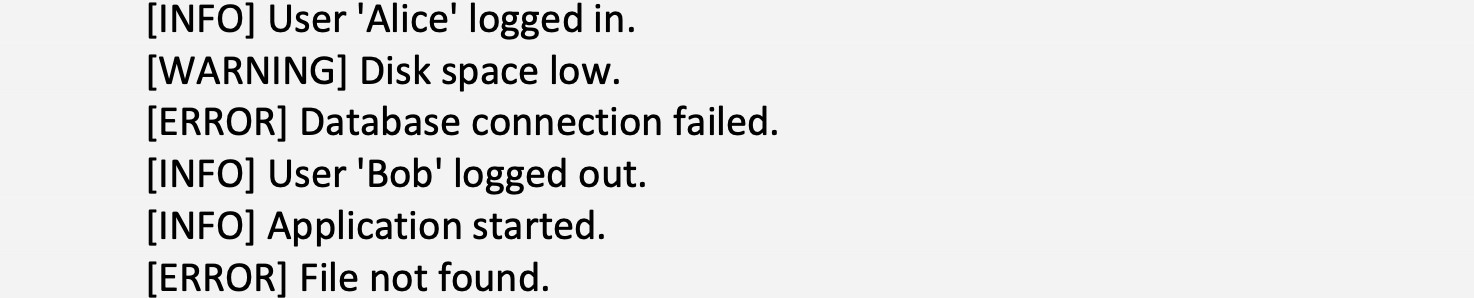
1. Use the datetime module to print the current date and time.
2. Use the random module to generate a random integer between 1 and 100 (inclusive).
3. Use the os module to list all files and directories in the current working directory.

###################

**Reading and Analyzing a Text File**

You have a log file (log.txt) from a server, and you need to extract some information from it.

**Tasks:**

1. Create a file named log.txt with the following content (or similar multi-line content):
2. 
3. Write a Python script to open log.txt in read mode.
4. Read the entire content of the file and print it to the console.
5. Count and print the number of lines in the file.
6. Count and print how many times the word "ERROR" appears in the file.

6.    ###################

**Analyzing Sales Data from CSV**

Your sales team has a CSV file with daily sales records, and they need to calculate total sales.

date,product,quantity,unit\_price

2024-01-01,Laptop,2,1200.00

2024-01-01,Mouse,5,25.50

2024-01-02,Keyboard,3,75.00

2024-01-02,T-Shirt,10,20.00

2024-01-03,Laptop,1,1200.00

**Tasks:**

1.     Create a CSV file named sales\_data.csv with the following content: Code snippet

2.     Write a Python script to open sales\_data.csv.

3.     Calculate the total revenue from all sales (quantity \* unit\_price for each row).

4.     Print the total revenue.

5.     Find and print the product that had the highest quantity sold.

**Pattern Matching**

You need to extract specific words or check for the presence of simple patterns in text.

**Tasks:**

1.     Define a string: text = "The quick brown fox jumps over the lazy dog."

2.     Write a Python script that uses the re module.

3.     Use re.search() to check if the word "quick" exists in the text. Print "Found 'quick'" if it

does, otherwise "Did not find 'quick'".

4.     Use re.findall() to find all occurrences of the letter "o" (case-sensitive) in the text. Print the

list of matches.

5.     Use re.search() with a regular expression to find if the text contains "fox" followed

immediately by "jumps". Print the match object if found.

**System Information with sys and os**

Your system administration team needs a quick script to gather basic information about the Python environment and the operating system.

**Tasks:**

1. Write a Python script that uses the sys module to print:  
o The Python version being used. ( sys.version() )   
o The current platform (e.g., 'linux', 'win32', 'darwin').  
o The maximum value an integer can take (if applicable, using sys.maxsize).

2. Use the os module to print:  
o The name of the operating system (e.g., 'posix', 'nt').  
o The current working directory.  
o The list of files and directories in the current working directory

**Student Grades and Courses**

Your school's administration wants to keep track of student grades across different courses.

**Tasks:**

1.     Create a dictionary called students where keys are student names (e.g., "Alice", "Bob") and values are dictionaries. Each inner dictionary should contain keys representing courses (e.g., "Math", "Science") and values representing their grades in those courses. Include at least three students and two courses per student.

2.     Print the students dictionary.

3.     Iterate through the students dictionary and print each student's name and their average

grade across all courses.

4.     Add a new course and grade for one of the existing students.

**Writing and Appending to a Text File**

You need to record sensor data and user activities to a file.

**Tasks:**

1.     Write a Python script that creates a new file named sensor\_data.txt.

2.     Write three lines of simulated sensor data to sensor\_data.txt, each on a new line (e.g.,

"Temperature: 25C", "Humidity: 60%", "Pressure: 1012hPa").

3.     Close the file.

4.     Re-open sensor\_data.txt in append mode.

5.     Append two new lines of simulated user activity to the file (e.g., "User 'Charlie' updated

settings", "System restart initiated").

6.     Print the entire content of sensor\_data.txt after both write and append operations to

verify.

**Error Prevention and Handling**

Your script needs to get numerical input from a user, but users sometimes enter text instead of numbers.

**Tasks:**

1.     Write a Python script that asks the user to enter an integer.

2.     Use a try-except block to **handle a ValueError**if the user enters non-integer input.

3.     If a ValueError occurs, print an informative error message (e.g., "Invalid input. Please

enter a whole number.").

4.     If the input is valid, print the square of the number.

5.     Include a finally block that prints "Input attempt completed." regardless of whether an

error occurred.

**Validating Input Formats**

Your system requires specific data formats for email addresses and phone numbers.

**Tasks:**

1.     Define a list of email addresses: emails = ["[user@example.com](mailto:user@example.com)", "invalid-email", "[test.user@domain.co.uk](mailto:test.user@domain.co.uk)", "another@.com"].

2.     Write a Python script.

3.     Use re.search() with a regular expression to validate each email address in the emails list. A

valid email should generally follow the pattern name@domain.extension.

4.     For each email, print whether it is "Valid" or "Invalid" based on your regex check.

5.     Define a list of phone numbers: phone\_numbers = ["123-456-7890", "1234567890", "abc-def-ghi"].

6.     Use re.findall() with a regular expression to extract all valid phone numbers.

**Measuring Script Execution Time with time**

You need to analyze the performance of different code snippets by measuring their execution time.

**Tasks:**

1.     Write a Python script.

2.     Use time.time() to record the start time.

3.     Implement a for loop that iterates 1,000,000 times and performs a simple arithmetic

operation (e.g., i \* 2).

4.     Record the end time using time.time().

5.     Calculate and print the total execution time of the loop in seconds.

6.     Use time.sleep() to pause your script for 2 seconds, and print a message before and

after the pause.

**Directory Operations with os**

Your program needs to organize files by creating new directories and checking their existence.

**Tasks:**

1.     Write a Python script.

2.     Define a directory name, e.g., new\_dir = "my\_temp\_folder".

3.     Use os.path.exists() to check if new\_dir already exists in the current working directory.

Print the result.

4.     If new\_dir does not exist, use os.mkdir() to create it. Print a confirmation message.

5.     Create a file inside my\_temp\_folder (e.g., my\_temp\_folder/test.txt). You can just open

and write a single line to it.

6.     Use os.listdir() to list the contents of my\_temp\_folder and print them.

**Processing Comma-Separated Values (CSV-like)**

You have a file (students.txt) containing student names and their scores, separated by commas. You need to process this data.

**Tasks:**

1.     Create a file named students.txt with the following content

Alice,85

Bob,92

Charlie,78

David,95

Eve,88

2.     Write a Python script to open students.txt in read mode.

3.     Read each line, parse the student's name and score, and store them in a list of

dictionaries (e.g., [{'name': 'Alice', 'score': 85}, ...]).

4.     Print the list of dictionaries.

5.     Calculate and print the average score of all students.

**Searching and Filtering File Content**

You have a large configuration file (config.txt) and you need to find specific settings. **Tasks:**

1.     Create a file named config.txt with the following content:

2.     Write a Python script to open config.txt in read mode.

3.     Read the file line by line.

4.     Print only those lines that contain an equals sign (=).

5.     Print only those lines that start with ADMIN\_ (ignoring case)

**Analyzing Sales Data from CSV**

Your sales team has a CSV file with daily sales records, and they need to calculate total sales.

**Tasks:**

1. Create a CSV file named sales\_data.csv with the following content: Code snippet

1. Write a Python script to open sales\_data.csv.
2. Calculate the total revenue from all sales (quantity \* unit\_price for each row).
3. Print the total revenue.
4. Find and print the product that had the highest quantity sold.

**File Not Found and Division by Zero**

You're building a simple calculator and a file reader, and both can encounter common errors.

**Tasks:**

1.     Write a Python script that attempts to open and read a file named non\_existent\_file.txt.

2.     Use a try-except block to **handle a FileNotFoundError**. If the file is not found, print "Error: The specified file does not exist."

3.     Within the same script, ask the user for two numbers.

4.     Use a try-except block to **handle a ZeroDivisionError**if the second number entered

is 0.

5.     If a ZeroDivisionError occurs, print "Error: Cannot divide by zero."

6.     If the division is successful, print the result of dividing the first number by the second.