Name : Ravi kumar yadav
E-mail : kumaryadavravi016@gmail.com
Assignment name : Data structure
Drive : drive
Github : Github

January 7, 2025

1 Assignment Files and Exceptional Handling.

2 Theory Questions

2.0.1 1) What is the difference between interpreted and compiled languages

(i) Interpreted languages (e.g., Python) are executed line-by-line by an interpreter. There is no intermediate machine code produced. The interpreter directly executes the code, which makes it easier to debug but typically slower.

(ii) Compiled languages (e.g., C, C++) are first converted into machine code by a compiler before they are executed. This machine code is directly run by the operating system, resulting in faster execution but less flexibility during development.

2.0.2 2) What is exception handling in Pytho?

Exception handling in Python refers to the process of managing runtime errors. It allows the program to continue running even if an error occurs. In Python, exceptions are handled using the try, except, else, and finally blocks.

2.0.3 3) What is the purpose of the finally block in exception handlin?

The finally block is used to execute code that must run regardless of whether an exception occurred or not. It's commonly used for cleanup actions like closing files or releasing resources.

2.0.4 4) What is logging in Python

Logging in Python refers to recording events that occur in a program, such as error messages or informational events, into a log file or output stream. Python's logging module is used to create logs and manage different log levels (e.g., DEBUG, INFO, WARNING).

2.0.5 5) What is the significance of the del method in Python?

The del method in Python is a destructor method, which is called when an object is about to be destroyed. It's useful for releasing resources (like closing files or database connections) when an object is no longer needed.

2.0.6 6) What is the difference between import and from ... import in Python?

(i) import module_name imports the entire module, so you must use the module name to access its functions (e.g., module_name.function()).

(ii) from module_name import function_name imports only the specified function or class, making it available directly without the module prefix (e.g., function_name()).

2.0.7 7) How can you handle multiple exceptions in Python?

We can handle multiple exceptions by using multiple except blocks, or by using a tuple to specify several exceptions in a single except block.

Exp : try:

code

except (TypeError, ValueError) as e:

handle errors

2.0.8 8) What is the purpose of the with statement when handling files in Python?

The with statement is used to handle files (or other resources) in a safe way. It ensures that the file is properly closed after the block is executed, even if an exception occurs. This simplifies file handling and eliminates the need to explicitly close the file.

2.0.9 9) What is the difference between multithreading and multiprocessing?

(i) Multithreading: Involves multiple threads within a single process. Threads share the same memory space, so communication between them is easier but prone to issues like race conditions.

(ii) Multiprocessing: Involves multiple processes running independently, each with its own memory space. It is more resource-intensive but avoids some problems associated with multithreading.

2.0.10 10) What are the advantages of using logging in a program?

(i) Tracks program execution

(ii)Helps debug and trace errors

(iii)Can be used to monitor performance or track key events

(iv) Makes it easier to identify issues in production systems

2.0.11 11) What is memory management in Python?

Memory management in Python involves the allocation and deallocation of memory automatically using Python's garbage collector. The memory is managed using reference counting and cyclic garbage collection to reclaim memory from unused objects.

2.0.12 12) What are the basic steps involved in exception handling in Python?

(i)try: Block of code that may raise an exception.

(ii)except: Block to handle the exception.

(iii) else: Block to run code if no exceptions occur.

(iv) finally: Block that runs regardless of exceptions.

2.0.13 13) Why is memory management important in Python?

Effective memory management is crucial to prevent memory leaks, optimize resource use, and ensure that a program doesn't consume more memory than necessary. Python automatically handles memory via garbage collection, but understanding it helps write more efficient and scalable programs.

2.0.14 14) What is the role of try and except in exception handling?

(i) The try block contains code that might throw an exception.

(ii) The except block catches and handles the exception if one is raised in the try block.

2.0.15 15) How does Python's garbage collection system work?

Python uses reference counting to track how many references exist for an object. If the reference count drops to zero, the object is deleted. Python also uses a cyclic garbage collector to handle cases where objects reference each other in cycles.

2.0.16 16) What is the purpose of the else block in exception handling?

The else block runs if no exceptions are raised in the try block. It is useful for code that should run only when no errors occur, helping to separate normal logic from error-handling logic.

2.0.17 17) What are the common logging levels in Python?

(i) DEBUG: Detailed information, typically useful only for diagnosing problems.

(ii) INFO: General information about the application's progress.

(iii)WARNING: Indications that something unexpected happened, but the program can continue.

(iv)ERROR: An error occurred, but it doesn't stop the program.

(v)CRITICAL: A severe error that likely causes the program to terminate.

2.0.18 18) What is the difference between os.fork() and multiprocessing in Python?

(i) os.fork() creates a new process by duplicating the current process. It is available only on UNIX-based systems.

(ii) multiprocessing is a high-level module that abstracts away the platform-specific details and provides a cross-platform approach for creating processes and managing inter-process communication.

2.0.19 19) What is the importance of closing a file in Python?

Closing a file ensures that all data is flushed to the file, and system resources (like file handles) are released. Not closing a file may result in data loss or memory/resource leaks.

2.0.20 20) What is the difference between file.read() and file.readline() in Python?

(i) file.read() reads the entire file as a single string.

(ii) file.readline() reads one line at a time from the file.

2.0.21 21) What is the logging module in Python used for?

The logging module is used to add logging to a Python program. It allows you to log messages at various severity levels, and these logs can be saved to a file, displayed in the console, or sent to other destinations.

2.0.22 22) What is the os module in Python used for in file handling?

The os module provides functions to interact with the file system, such as creating, deleting, and manipulating files and directories. It also provides functionality for checking file existence, file permissions, and path manipulations

2.0.23 23) What are the challenges associated with memory management in Python?

(i) Python's automatic garbage collection can lead to inefficient memory usage in some cases.

(ii) Cycles of references (objects referring to each other) may not be collected immediately.

(iii) Developers need to ensure memory is freed correctly, especially for large objects or external resources.

2.0.24 24) How do you raise an exception manually in Python?

We can raise an exception using the raise keyword:

Exp : raise ValueError("This is a custom exception message")

2.0.25 25) Why is it important to use multithreading in certain applications?

Multithreading is used in applications that require concurrent execution of tasks, such as I/O-bound operations (e.g., web servers, database handling) or for improving the responsiveness of an application. It allows for better utilization of CPU resources in some scenarios.

3 Practical questions

3.0.1 1) How can you open a file for writing in Python and write a string to it?

[74]:
```
file = open('file.txt','w')
file.write('Hi my name is \'Ravi kumar yadav\'  ')
file.close()
```

3.0.2 2) Write a Python program to read the contents of a file and print each line

[78]:
```
file = open('file.txt','r')
for i in file:
    print(i)
```

```
Hi my name is 'Ravi kumar yadav'
```

3.0.3 3) How would you handle a case where the file doesn't exist while trying to open it for reading?

[27]:
```
try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("The file does not exis.")
```

```
The file does not exis.
```

3.0.4 4) Write a Python script that reads from one file and writes its content to another file?

[29]:
```
try:
    with open('source.txt', 'r') as source_file:
        content = source_file.read()
    with open('destination.txt', 'w') as dest_file:
        dest_file.write(content)
```

```
except FileNotFoundError:
    print("Source file does not exist.")
```

```
Source file does not exist.
```

3.0.5 5) How would you catch and handle division by zero error in Python?

[31]:
```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

```
Cannot divide by zero.
```

3.0.6 6) Write a Python program that logs an error message to a log file when a division by zero exception occurs?

[19]:
```
import logging

# Set up basic configuration for logging
logging.basicConfig(filename='error_log.log', level=logging.ERROR,
                    format='%(asctime)s - %(levelname)s - %(message)s')

try:
    # Code that might cause a division by zero
    result = 10 / 0
except ZeroDivisionError as e:
    # Log the error message when a division by zero occurs
    logging.error("Division by zero error occurred: %s", e)

print("Error has been logged.")
```

```
Error has been logged.
```

3.0.7 7) How do you log information at different levels (INFO, ERROR, WARNING) in Python using the logging module

[1]:
```
import logging

# Set up basic configuration for logging
logging.basicConfig(filename='error_log.log', level=logging.ERROR,
                    format='%(asctime)s - %(levelname)s - %(message)s')

try:
    # Code that might cause a division by zero
    result = 10 / 0
except ZeroDivisionError as e:
```

```
    # Log the error message when a division by zero occurs
    logging.error("Division by zero error occurred: %s", e)

print("Error has been logged.")
```

```
Error has been logged.
```

3.0.8 8) Write a program to handle a file opening error using exception handling?

[3]:
```
try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("File not found. Please check the file pat.")
```

```
File not found. Please check the file pat.
```

3.0.9 9) How can you read a file line by line and store its content in a list in Python?

[5]:
```
lines = []
try:
    with open('file.txt', 'r') as file:
        for line in file:
            lines.append(line.strip())  # Here might be an unmatched parenthesis
except FileNotFoundError:
    print("The file does not exist.")
print(lines)
```

```
["Hi my name is 'Ravi kumar yadav'"]
```

3.0.10 10) How can you append data to an existing file in Python?

[9]:
```
with open('file.txt', 'a') as file:
    file.write("New data to append.\n")
```

3.0.11 11) Write a Python program that uses a try-except block to handle an error when attempting to access a dictionary key that doesn't exist.t

[15]:
```
my_dict = {"a": 1, "b": 2}
try:
    value = my_dict["c"]
except KeyError:
    print("Key does not exist in the dictionary.")
```

```
Key does not exist in the dictionary.
```

3.0.12 12) Write a program that demonstrates using multiple except blocks to handle different types of exceptions.?

[13]:
```
try:
    result = 10 / 0
    my_dict = {"a": 1}
    value = my_dict["b"]
except ZeroDivisionError:
    print("Cannot divide by zero.")
except KeyError:
    print("Key not found in dictionary.")
```

```
Cannot divide by zero.
```

3.0.13 13) How would you check if a file exists before attempting to read it in Python?

[1]:
```
import os

if os.path.exists('file.txt'):
    with open('file.txt', 'r') as file:
        content = file.read()
else:
    print("The file does not exist.")
```

3.0.14 14) Write a program that uses the logging module to log both informational and error messages.

[3]:
```
import logging

logging.basicConfig(filename='app.log', level=logging.DEBUG)

logging.info("This is an informational message.")
try:
    1 / 0
except ZeroDivisionError:
    logging.error("Error: Division by zero occurred.")
```

3.0.15 15) Write a Python program that prints the content of a file and handles the case when the file is empty.

[7]:
```
try:
    with open('file.txt', 'r') as file:
        content = file.read()
        if not content:
            print("The file is empty.")
        else:
            print(content)
```

```
except FileNotFoundError:
    print("The file does not xist.")
```

```
Hi my name is 'Ravi kumar yadav'
```

3.0.16 16) Demonstrate how to use memory profiling to check the memory usage of a small program.

[1]:
```
from memory_profiler import profile
@profile
def my_function():
    a = [1] * (10 ** 6)  # List of 1 million integers
    b = [2] * (2 * 10 ** 7)  # List of 20 million integers
    del b  # Free up memory by deleting b
    return a

if __name__ == "__main__":
    my_function()
```

```
ERROR: Could not find file
C:\Users\kumar\AppData\Local\Temp\ipykernel_5548\2819384872.py
```

3.0.17 17) Write a Python program to create and write a list of numbers to a file, one number per line.

[15]:
```
numbers = [1, 2, 3, 4, 5]
with open('numbers.txt', 'w') as file:
    for number in numbers:
        file.write(f"{number}\n")
```

3.0.18 18) How would you implement a basic logging setup that logs to a file with rotation after 1MB?

[11]:
```
import logging
from logging.handlers import RotatingFileHandler

logger = logging.getLogger('my_logger')
handler = RotatingFileHandler('my_log.log', maxBytes=1e6, backupCount=3)
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

logger.info("This is an info message.")
```

3.0.19 19) Write a program that handles both IndexError and KeyError using a try-except block.

[11]:
```
my_list = [1, 2, 3]
my_dict = {"a": 1}

try:
    value = my_list[5]
    value = my_dict["b"]
except IndexError:
    print("Index is out of range.")
except KeyError:
    print("Key not found in dictonary.")
```

```
Index is out of range.
```

3.0.20 20) How would you open a file and read its contents using a context manager in Python?

[9]:
```
with open('file.txt', 'r') as file:
    content = file.read()
print(content)
```

```
Hi my name is 'Ravi kumar yadav'
```

3.0.21 21) Write a Python program that reads a file and prints the number of occurrences of a specific word.

[9]:
```
word_to_find = "Python"
word_count = 0

try:
    with open('file.txt', 'r') as file:
        for line in file:
            word_count += line.lower().count(word_to_find.lower())
except FileNotFoundError:
    print("The file does not exist.")

print(f"The word '{word_to_find}' appears {word_count} times.")
```

```
The word 'Python' appears 0 times.
```

3.0.22 22) How can you check if a file is empty before attempting to read its contents?

[5]:
```
import os

if os.stat('file.txt').st_size == 0:
    print("The file is empty.")
```

```
else:
    with open('file.txt', 'r') as file:
        content = file.read()
        print(content)
```

```
Hi my name is 'Ravi kumar yadav'
```

3.0.23 24) Write a Python program that writes to a log file when an error occurs during file handling.

[3]:
```
import logging

logging.basicConfig(filename='file_error.log', level=logging.ERROR)

try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError as e:
    logging.error(f"Error occurred while opening file: {e}")
```