pxhwb5k3n

January 15, 2025

Files, Exceptional handeling, Logging, Memory management.

---

1. What is the difference between interpreted and compiled languages?

- Interpreted -> An Interpreted Language is a type of programming language where the code is compiled on the fly each time the program is run.
- Compiled -> compiled languages here the code is compiled only once into machine code.
- Compiled language follows at least two levels to get from source code to execution. Interpreted language follows one step to get from source code to execution. A compiled language is converted into machine code so that the processor can execute it.

2. What is exception handling in Python?

- It means to handel any error which occures in doing programming or coding. exceptional handeling prevent any exception code to give error and stop the further running of code.

3. What is the purpose of the finally block in exception handling?

- Finally block gives the final output of the code if code has any error or not finally block code will run always.

4. What is logging in Python?

- Python logging is a module that allows you to track events that occur while your program is running. You can use logging to record information about errors, warnings, and other events that occur during program execution. And logging is a useful tool for debugging, troubleshooting, and monitoring your program.

5. What is the significance of the **del** method in Python?

- **del** method helps to remove or delete the function or any other code which is mentioned in **del** method.

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

- imoprt method imports the entire library functions which is related to that library what we called with import.
- from … import does import only a particular function from the library what we called not the entire library's functions.

7. How can you handle multiple exceptions in Python?

- Python allows you to catch multiple exceptions in a single 'except' block by specifying them as a tuple. This feature is useful when different exceptions require similar handling logic.

In this case, if either 'ExceptionType1' or 'ExceptionType2' is raised, the code within the 'except' block will be executed.

8. What is the purpose of the with statement when handling files in Python

- The with statement in Python is used to simplify the management of resources that need to be acquired and released in a specific order. It provides a way to wrap the execution of a block of code with methods defined by a context manager.

9. What is the difference between multithreading and multiprocessing?

- Multithreading refers to the ability of a processor to execute multiple threads concurrently, where each thread runs a process.
- Multiprocessing refers to the ability of a system to run multiple processors in parallel, where each processor can run one or more threads.

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

- It is also useful to log successful requests so we can have a clear idea of how users work with the application. But this goes beyond that. Logs are also useful to detect common mistakes users make, as well as for security purposes.

11. What is memory management in Python?

- Processors had multiple cores and threads by default all programm or work perfors only one core and one thread, memory management helps to use the other cores and threads it helps to save the time.

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

- basic steps are, use Try: method to write the exceptional code into it and then use except block to find the which type of error it could occurs. we can also use the finally block but it's act like a normal code to run.

13. Why is memory management important in Python?

- memory management helps to save the time because memory management use multiple cores and threads which helps to perform the task in few times than actual time. it's also helps to use multiple cores and threads in processors of CPU.

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

- Try block is used for writing the exceptional code which code is doubtful code, it runs the code and sees the given code is perfect or had any errors into it.
- except block helps to get information about the if there is any error it help to find that error and type of that error and does not stops the further code to run.

15. How does Python's garbage collection system work?

- Garbage collection is the process of releasing memory when the object is no longer in use. This system destroys the unused object and reuses its memory slot for new objects.

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

- else block helps to procces the code if there is no error in given code to try block to try the code if there is any error or not. If not then else block code excutes…

17. What are the common logging levels in Python?

- common logging levels are :
- Debug
- Info
- Warning
- Error
- Critical

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

- The fork method is only supported on POSIX-based systems like Linux and macOS (not Windows). fork used for running the code or performing the code faster than multiprocessing this is the only difference between fork method and multiprocessing.

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

- closing a file is important because other can not perform any changes to the file which is opened, mainly it's important as the safety purposes.

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

- file.read() helps to read the entire file in one go, but on the other hand file.readline() helps to read the file line by line instead of entire file in one go.

21. What is the logging module in Python used for?

- Python logging is a module that allows you to track events that occur while your program is running. You can use logging to record information about errors, warnings, and other events that occur during program execution. And logging is a useful tool for debugging, troubleshooting, and monitoring your program.

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

- Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

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

- Memory management performs the program by using multiple cores and threads that's why it's helps to do perform the task faster but because it perform the codes in different cores that's why there is no sequence of output happens.

24. How do you raise an exception manually in Python?

- As a Python developer you can choose to throw an exception if a condition occurs. To throw (or raise) an exception, use the raise keyword.

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

- Multithreading enhances scalability by allowing a program to take advantage of multiple CPU cores or processors. Tasks can be divided into threads that can execute concurrently, enabling better utilization of available hardware resources and improving overall system performance as workload increases.

[ ]:
```
#1.  How can you open a file for writing in Python and write a string to it?

file = open("file.txt", "w")
file.write("example string")
file.close()
```

[ ]:
```
#2.  Write a Python program to read the contents of a file and print each line?

file = open("file.txt", "r")
print(file.read())
```

[2]:
```
#3.  How would you handle a case where the file doesn't exist while trying to␣
 ↪open it for reading?

try:
  with open("file1.txt", "r") as file:
    file
except Exception as e:
  print(e)
else:
  print(file.read())
```

```
[Errno 2] No such file or directory: 'file1.txt'
```

[ ]:
```
#4.  Write a Python script that reads from one file and writes its content to␣
 ↪another file.

file = open("file.txt", "w")
file.write("line of the first file.")

file = open("file.txt", "r")
read_file = file.read()

def create_file(x):
    file2 = open("newfile.txt", "w")
    file2.write(x)

create_file(read_file)
```

[6]:
```
#5.  How would you catch and handle division by zero error in Python?

try:
  10/0
except ZeroDivisionError as e:
  print("there is a error =",e)
```

```
there is a error = division by zero
```

[ ]:
```
#6.  Write a Python program that logs an error message to a log file when a␣
 ↪division by zero exception occurs.

import logging

logging.basicConfig(filename= "file1.log")

try:
    10/0
except Exception as e:
    logging.error(f"this is the error occours when running the code = {e}")
```

[9]:
```
from sre_constants import INFO
#7.  How do you log information at different levels (INFO, ERROR, WARNING) in␣
 ↪Python using the logging module?

import logging

logging.basicConfig(filename = "log_file.log")

logging.info("this is the information about the log file.")
logging.error("this is the error occurs when running the code.")
logging.warning("it's a warning to log file.")
```

```
ERROR:root:this is the error occurs when running the code.
WARNING:root:it's a warning to log file.
```

[10]:
```
#8.  Write a program to handle a file opening error using exception handling.

try:
    with open("file.txt") as file:
        file
except Exception as e:
    print(f"error = {e}")
```

```
error = [Errno 2] No such file or directory: 'file.txt'
```

[ ]:
```
#9.  How can you read a file line by line and store its content in a list in␣
 ↪Python?

file = open("test_file.txt", "r")

file_list = []
for i in range(3):
    a = file.readline()
    file_list.append(a)
```

```
print(file_list)
```

[16]:
```
#10  How can you append data to an existing file in Python?

file = open("test_file.txt", "a")

file.write("this is the fourth line.")
file.close()
```

[14]:
```
#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.

new_dict = {"key1" : 123, "key2" : 456}

try:
    print(new_dict["key3"])

except Exception as e:
    print(f"there is a error {e}")
```

```
there is a error 'key3'
```

[15]:
```
#12.  Write a program that demonstrates using multiple except blocks to handle␣
 ↪different types of exceptions.

new_dict = {"key1" : 123, "key2" : 456}

try:
    print(new_dict["key3"])

except ZeroDivisionError as e:
    print(f"there is a error {e}")

except KeyError as e:
    print(f"there is a error {e}")
```

```
there is a error 'key3'
```

[17]:
```
#13.  How would you check if a file exists before attempting to read it in␣
 ↪Python?

from pathlib import Path

print(Path("file.txt").exists())
```

```
False
```

[18]:
```
#14.  Write a program that uses the logging module to log both informational␣
 ↪and error messages.

import logging

import logging

logging.basicConfig(filename = "file.log", level = logging.INFO)

logging.info("this is the information about log file.")
logging.error("error occcurs when running the file.")
```

```
ERROR:root:error occcurs when running the file.
```

[20]:
```
#15.  Write a Python program that prints the content of a file and handles the␣
 ↪case when the file is empty.

try:
  file = open("test.txt", "r")
  print(file.read())
except Exception as e :
  print(f"error = {e}")
```

```
error = [Errno 2] No such file or directory: 'test.txt'
```

[22]:
```
#16.  Demonstrate how to use memory profiling to check the memory usage of a␣
 ↪small program.

from memory_profiler import profile

@profile

def my_func():
    x = [x for x in range(0, 100)]
    y = [y*100 for y in range(0, 150)]
    del x
    return y

if __name__ == '__main__':
    my_func()
```

[24]:
```
#17.  Write a Python program to create and write a list of numbers to a file,␣
 ↪one number per line.

list = [1, 2, 3, 4, 5]
with open("list_file.txt", "w") as file:
    for i in list:
```

```
        file.write(f"{i}\n")
```

[ ]:
```
#18.  How would you implement a basic logging setup that logs to a file with␣
 ↪rotation after 1MB?

import logging
import time

from logging.handlers import RotatingFileHandler

def create_rotating_handler(Path):
    logger = logging.getLogger("Rotating Log")
    logger.setLevel(logging.INFO)

    handler = RotatingFileHandler(Path, maxBytes = 1)
    logger.addHandler(handler)

    for i in range(10):
        logger.info("this is test log line %s" %i)
        time.sleep(1)

if __name__ == "__main__":
    log_file = "test_log"
    create_rotating_handler(log_file)
```

[25]:
```
#19.  Write a program that handles both IndexError and KeyError using a␣
 ↪try-except block.

new_list = [1, 2, 3, 4, 5]
new_dict = {"key1" : 1, "key2" : 2, "key3" : 3}

try:
    print(new_list[5])

except IndexError as i:
    print(f"there is error = {i}")

try:
    print(new_dict["key4"])
except KeyError as k:
    print(f"there is error = {k}")
```

```
there is error = list index out of range
there is error = 'key4'
```

[ ]:
```
#20.  How would you open a file and read its contents using a context manager␣
 ↪in Python?
```

```
file = open("file.txt", "r")
file.read()
```

[26]:
```
#21.  Write a Python program that reads a file and prints the number of␣
 ↪occurrences of a specific word.

file = open("test_file.txt", "r")

a = file.read()
print(a.count("line"))
```

```
2
```

[27]:
```
#22.  How can you check if a file is empty before attempting to read its␣
 ↪contents?

from pathlib import Path

print(Path("file.txt").exists())
```

```
False
```

[28]:
```
#23.  Write a Python program that writes to a log file when an error occurs␣
 ↪during file handling.

import logging

logging.basicConfig(filename = "file.log")

try:
    with open("file.txt", "r") as file:
        file
except Exception as e:
    logging.error(f"error is {e}")
```

```
ERROR:root:error is [Errno 2] No such file or directory: 'file.txt'
```