THEORY ASSIGNMENTS

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

Ans: A compiled language is a programming language whose implementations are typically compilers and not interpreters. An interpreted language is a programming language whose implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions.

2. What is exception handling in Python?

Ans: Exception handling in Python is a mechanism to gracefully manage errors that occur during the execution of a program. These errors, known as exceptions, can disrupt the normal flow of the program if not handled properly. Exception handling allows the program to continue running, preventing abrupt termination and providing a more robust user experience.

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

Ans: The purpose of the finally block in exception handling is to ensure that certain code, such as cleanup operations or resource releases, executes regardless of whether an exception is thrown or caught in the try block. It guarantees that critical tasks, like closing files or releasing connections, will be performed, even if an exception occurs and the program flow is interrupted.

4. What is logging in Python?

Ans: Logging is a means of tracking events that happen when some software runs. Logging is important for software developing, debugging, and running. If you don't have any logging record and your program crashes, there are very few chances that you detect the cause of the problem. And if you detect the cause, it will consume a lot of time. With logging, you can leave a trail of breadcrumbs so that if something goes wrong, we can determine the cause of the problem.

5. What is the significance of the __del__ method in Python?

Ans: The __del__ method in Python, often referred to as a destructor, is a special method called when an object is about to be destroyed. It provides an opportunity to perform cleanup actions, such as releasing external resources or finalizing operations before the object's memory is reclaimed. However, its behaviour is not always predictable.

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

Ans: import module name:

This statement imports the entire module, making it accessible through its namespace. To use functions or classes within the module, one must prefix them with the module name (e.g., module_name. function_name).

From module name import element:

This statement imports specific elements (functions, classes, variables) directly into the current namespace. These elements can be used without the module name prefix (e.g., function_name). It is also possible to import all elements using the asterisk (*) symbol, but this is generally discouraged due to potential naming conflicts.

7. How can you handle multiple exceptions in Python?

Ans: Multiple exceptions in Python can be handled using several approaches:

- Using separate except blocks: This involves writing different except blocks for each type of exception that needs to be handled. This is useful when different exceptions require different handling logic.
- Using a single except block with a tuple of exceptions. This approach allow handling multiple exceptions with the same logic in a single except block.

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

Ans: The with statement in Python serves to simplify file handling by ensuring that files are properly opened and closed, even if errors occur. It acts as a context manager, automatically managing resources and handling exceptions. When working with files, using with guarantees that the file will be closed after the block of code within the with statement is executed, preventing resource leaks and potential data corruption.

9. What is the difference between multithreading and multiprocessing?

Ans: Multithreading and multiprocessing are both techniques to run multiple tasks concurrently, but they differ in how they achieve this. Multithreading creates multiple threads within a single process, allowing for concurrent execution within that process, while multiprocessing creates multiple processes, each with its own resources, enabling parallel execution across multiple processors.

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

Ans: Logging offers significant advantages in software development, including improved debugging, easier troubleshooting, enhanced system observability, and better communication between developers and administrators. It provides a record of events, helping identify issues, understand system behaviour, and optimize performance.

11. What is memory management in Python?

Ans: Memory management in Python involves the allocation and deallocation of memory resources for objects. Python utilizes a private heap to store objects and data structures. The Python memory manager handles this private heap internally, featuring components for various dynamic storage management aspects like sharing, segmentation, reallocation, and caching. At the lowest level, a raw memory allocator interacts with the operating system's memory manager to ensure sufficient space in the private heap for Python-related data.

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

Ans: In Python, exceptions are caught and handled using the 'try' and 'except' block. 'try' contains the code segment which is susceptible to error, while 'except' is where the program should jump in case an exception occurs. You can use multiple 'except' blocks for handling different types of exceptions.

13. Why is memory management important in Python?

Ans: Memory management is important in Python because it ensures efficient use of system resources, prevents memory leaks, and ultimately contributes to the stability and performance of applications. Python employs automatic memory management through a mechanism called garbage collection, which handles the allocation and deallocation of memory for objects.

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

Ans: In exception handling, try and except blocks work together to gracefully handle errors or exceptions that may occur during code execution. The try block contains the code that might potentially raise an exception, and the except block contains the code that will be executed if an exception is raised within the try block. This allows the program to continue running instead of crashing when an error occurs.

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

Ans: Python uses a hybrid approach to garbage collection: reference counting and generational garbage collection. Reference counting efficiently handles most cases where an object's reference count reaches zero, indicating no more active references. However, reference cycles, where objects refer to each other, can prevent reference counts from dropping to zero, necessitating the generational garbage collector.

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

Ans: The else block in exception handling, often used in try...except...else structures, executes only when no exceptions are raised within the try block. It provides a way to execute code that's intended to run when the try block executes successfully, effectively separating normal execution from exception handling.

17. What are the common logging levels in Python?

Ans: There are six levels for logging in Python; each level is associated with an integer that indicates the log severity: NOTSET=0, DEBUG=10, INFO=20, WARN=30, ERROR=40, and CRITICAL=50.

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

Ans: The os.fork() system call and the multiprocessing module in Python both enable the creation of new processes, but they differ significantly in their approach, portability, and use cases.

os.fork() creates a new process that is a nearly exact copy of the calling process, including its memory space and file descriptors. It's a low-level system call, primarily available on Unix-like systems, and directly interacts with the operating system.

multiprocessing, on the other hand, is a higher-level module in Python that provides a more portable and feature-rich way to manage processes. It can use different start methods (like "fork," "spawn," or "forkserver") depending on the operating system and configuration, offering greater flexibility.

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

Ans: Closing files in Python is an essential practice that helps maintain data integrity, prevent resource leaks, and ensure the reliability of your applications. By mastering file handling techniques, you can write more robust and efficient Python code that effectively manages file resources.

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

Ans: In Python, the `read()` method is used to read a specified number of characters from a file or input stream, while the `readline()` method is used to read a single line from a file or input stream. The `read()` method will read the entire content of the file or stream if no argument is provided, returning a string.

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

The logging module in Python is used for recording events and debugging issues during application execution. It provides a flexible system for logging messages, including errors, warnings, and informational messages, to various output destinations like files or the console.

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

Ans: 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?

Ans: Memory management problems: The basic problem in managing memory is knowing when to keep the data it contains, and when to throw it away so that the memory can be reused. This sounds easy, but is, in fact, such a hard problem that it is an entire field of study in its own right.

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

Ans: 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?

Ans: Multithreading is important in applications that benefit from concurrent execution of tasks, improved responsiveness, and efficient resource utilization. By breaking down a program into smaller, independent threads, applications can perform multiple operations simultaneously, leading to faster execution, smoother user interfaces, and better utilization of CPU cores.