Q1. Describe three applications for exception processing.

Ans: exception processing is a mechanism used in python to handle unexpected errors or exceptional conditions that occur during program execution. Following are the 3 applications of it.

1. Error handling: when a program encounters an error, it throws an exception, which can be caught and handled by exception handling.
2. Resource management: when a program uses resources such as memory, files. It is important to ensure that these resources are properly allocated and released. Exception processing can be used to handle exceptions that occur during resources allocation or deallocations.
3. Input validation: when a program accepts input from a user. It is important to ensure that the input is valid and does not cause any unexpected behaviour. Exception processing can be used to handle exceptions that occur when input validation fails.

Q2. What happens if you don't do something extra to treat an exception?

Ans: if we don’t handle exception properly, the program will terminate abnormally, and error message will be displayed to the user.

Q3. What are your options for recovering from an exception in your script?

Ans: Catch the exception and handle it using “try” and “except” block. we could print an error message, log the error, or try again with a different approach.

Q4. Describe two methods for triggering exceptions in your script.

Ans: 1. Raise an exception: we can use the ‘raise’ statement to explicitly raise an exception in your code. For example you can raise a “value error” exception when function argument is invalid.

2. let python raise an exception implicitly: python will automatically raise an exception when it encounters an error that it cannot handle. For example, if you try to open a file that doesnot exist, python will raise a “FileNotFoundError”.

Q5. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.

Ans: 1. The “finally” block is used to define a block of code that will be executed regardless of whether an exception is thrown or not. This allows to ensure that certain actions are always executed, such as closing resources like database connections, network sockets and file.

2. context manager: context manager are objects that define a runtime context in which certain operations can be performed. They allow you to allocate and release resources in a safe and controlled way. And resources are properly cleaned up when they are no longer needed. A context manager is typically created using the “with” statement in python.

Example: with open(“file.txt”, “w”) as f:

f.write(“hi there”)