Q1. Describe three applications for exception processing.

Ans : Exception Handling is a mechanism to handle runtime errors such as

1. IndexError

The IndexError is thrown when trying to access an item at an invalid index.

>>> L1=[1,2,3]

>>> L1[3]

Traceback (most recent call last):

File "<pyshell#18>", line 1, in <module>

L1[3]

IndexError: list index out of range

1. ModuleNotFoundError

The ModuleNotFoundError is thrown when a module could not be found.

>>> import notamodule

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

import notamodule

ModuleNotFoundError: No module named 'notamodule'

1. KeyError

The KeyError is thrown when a key is not found.

>>> D1={'1':"aa", '2':"bb", '3':"cc"}

>>> D1['4']

Traceback (most recent call last):

File "<pyshell#15>", line 1, in <module>

D1['4']

KeyError: '4'

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

Ans : When an exception occurred, if you don't handle it, the program terminates abruptly and the code past the line that caused the exception will not get executed.

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

Ans : The except block is executed when an exception occurs within the try block. The optional else block is executed only if there were no exceptions after try and before finally . The finally block contains instructions that are always executed at the end, regardless of whether exceptions occurred in the try block.

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

Ans : A direct logic is followed to catch exceptions in Python. When an exception occurs, the Python interpreter stops the current process. It is handled by passing through the calling process. If not, the program will crash.

For instance, a Python program has a function X that calls function Y, which in turn calls function Z. If an exception exists in function Z but is not handled within Z, the exception passes to Y and then to X. Simply, it will have a domino effect.

An unhandled exception displays an error message and the program suddenly crashes. To avoid such a scenario, there are two methods to handle Python exceptions:

Try – This method catches the exceptions raised by the program

Raise – Triggers an exception manually using custom exceptions

Let’s start with the try statement to handle exceptions. Place the critical operation that can raise an exception inside the try clause. On the other hand, place the code that handles the exceptions in the except clause.

Developers may choose what operations to perform once it catches these exceptions. Take a look at the sample code below:

import sys

list = [‘x’, 1e-15, 5]

for result in list:

try:

print(“The result is”, result)

y = 1/int(result)

break

except:

print(“Whew!”, sys.exc\_info()[0], “occurred.”)

print(“Next input please.”)

print()

print(“The answer of”, result, “is”, y)

The program has an array called list with three elements. Next, the line that causes an exception is placed inside the try block. If there are no exceptions, the except block will skip, and the logic flow will continue until the last element. However, if an exception occurs, the except block will catch it.

The output:

The result is x

Whew! <class ‘ValueError’> occurred.

Next input, please.

The result is 1e-15

Whew! <class ‘ZeroDivisionError’> occurred.

Next input, please.

The result is 5

The answer to 5 is 0.2

To print the name of the exception, we use the exc\_info()function inside the sys module. The first two elements cause exceptions since an integer can’t be divided with string and a zero value. Hence, it raised ValueError and ZeroDivisionError exceptions.

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

Ans : Logging, closing some open connection, closing open files.