Exception Handling Theory Questions

1.What is an exception?

In Python, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an exception occurs, the program stops execution and raises an error message to indicate what went wrong.

Exceptions can occur for a variety of reasons, such as attempting to divide by zero, trying to access a file that doesn't exist, or passing an invalid argument to a function. In general, exceptions are raised whenever a program encounters an error that it can't handle itself.

However, Python provides a mechanism for handling exceptions called exception handling. By using try and except blocks, you can catch and handle exceptions in a way that allows your program to continue running without crashing.

2. Can you explain the difference between errors and exceptions?

errors	exceptions
1. An error is a problem that occurs at a low level within the Python interpreter or operating system, and usually results in the program crashing or stopping abruptly.	1. An exception, on the other hand, is an error that occurs during the execution of a program that disrupts the normal flow of instructions.
2. Examples of errors in Python include syntax errors, indentation errors, and name errors.	2. Exceptions are raised by the program itself when it encounters a problem that it can't handle, such as trying to divide by zero, accessing a file that doesn't exist, or passing an invalid argument to a function.

3. How can you ensure that a certain code block runs no matter whether there's an exception or not?

To ensure that a certain code block runs no matter whether there's an exception or not, you can use a finally block. The finally block is executed after the try block and any associated except blocks, regardless of whether an exception was raised or not.

4. Can you give me some examples of built-in exceptions?

Here are some examples of built-in exceptions in Python

1. TypeError: Raised when an operation or function is applied to an object of inappropriate type.
2. ValueError: Raised when a function receives an argument of correct type but inappropriate value.
3. NameError: Raised when a local or global name is not found.
4. IndexError: Raised when an index is not found in a sequence.
5. KeyError: Raised when a key is not found in a dictionary.
6. ZeroDivisionError: Raised when the second argument of a division or modulo operation is zero.
7. FileNotFoundError: Raised when a file or directory is requested but doesn't exist.
8. OverflowError: Raised when a calculation exceeds the maximum limit for a numeric type.

These are just a few examples of the many built-in exceptions available in Python.

5. How do you handle exceptions with try/except/finally?

1. In Python, you can handle exceptions using the try/except/finally block. Here's how it works:First, you put the code that may raise an exception inside a try block.
2. If an exception is raised in the try block, the corresponding except block is executed. You can specify the type of exception to catch by adding the exception type in parentheses after the except keyword.
3. After the except block, you can optionally include a finally block. This block is executed no matter what, whether an exception was raised or not.

6. "Every syntax error is an exception but every exception cannot be a syntax error". Justify the Statement.

The statement "Every syntax error is an exception but every exception cannot be a syntax error" is true in Python.

In Python, a syntax error is a specific type of exception that occurs when the interpreter encounters code that violates the language syntax rules. Syntax errors are detected during the parsing phase, and they prevent the code from being executed. Examples of syntax errors include misspelled keywords, missing colons, mismatched parentheses, and invalid variable names.

On the other hand, exceptions are a more general category of errors that occur during program execution. They can be caused by a variety of factors, including user input, system resources, and external dependencies. Examples of exceptions in Python include TypeError, ValueError, ZeroDivisionError, and IOError.

Therefore, while every syntax error is an exception because it is an error that occurs during program execution, not every exception is a syntax error because exceptions can occur for many reasons other than violating syntax rules.

7. What is the use of raise statement?

In Python, the raise statement is used to raise an exception manually. This means that you can define your own custom exception and raise it under certain conditions, or you can re-raise an existing exception that has been caught in a try-except block.

8. What is the use of else block in exception handling?

The else block in exception handling is optional and it is executed only if no exception is raised in the try block. If any exception is raised, the code in the else block will not be executed. The else block is generally used for adding code that should be executed if the try block runs successfully without any exceptions.

9. What is the syntax for try and except block?

The basic syntax for the `try` and `except` block in Python is:

```
try:
  block of code where the error can occur
except ExceptionName:
   block of code to be executed when ExceptionName occurs
```

Here, the `try` block contains the code that might raise an exception, and the `except` block contains the code that will handle the exception if it occurs. If no exception occurs, the `except` block will be skipped.

10. What is the purpose of finally block?

The finally block in Python is used to specify a block of code that will be executed no matter whether an exception is raised or not.

The purpose of the finally block is to ensure that certain actions are performed regardless of whether an exception is raised or not. For example, if a file is opened in the try block, the finally block can be used to close the file, so that resources are freed up and the file is not left open.

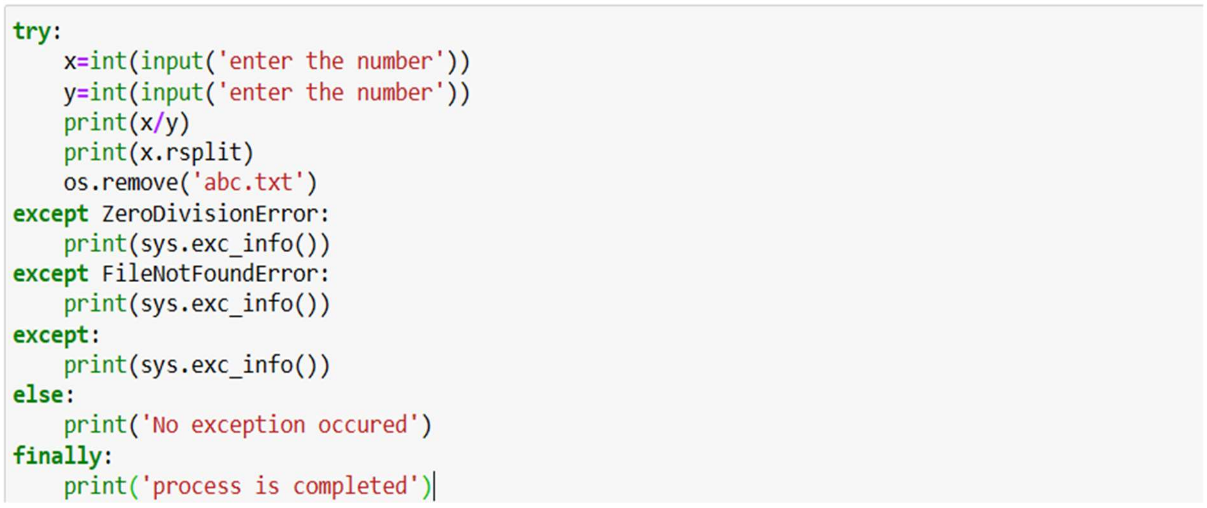
11. Can you explain the difference between try-except and try-finally block?

Both try-except and try-finally blocks are used in exception handling in Python. However, their purposes are different:

1. try-except: This block is used to catch and handle specific exceptions that may occur in a try block. If an exception occurs, the code in the except block will be executed, allowing the program to handle the exception and continue running.
2. try-finally: This block is used to execute cleanup code, whether an exception occurs or not. The code in the try block is executed, and then the code in the finally block is executed, allowing the program to release resources, close files, or perform other necessary tasks.

12. Can you give example of using multiple except block in try-except block?

Here is an example of using multiple except blocks in a try-except block:



```
try:
    x=int(input('enter the number'))
    y=int(input('enter the number'))
    print(x/y)
    print(x.rsplit)
    os.remove('abc.txt')
except ZeroDivisionError:
    print(sys.exc_info())
except FileNotFoundError:
    print(sys.exc_info())
except:
    print(sys.exc_info())
else:
    print('No exception occured')
finally:
    print('process is completed')
```

In this example, the code inside the try block takes two integer inputs from the user and performs a division operation. If any exceptions occur during the execution of the try block, the corresponding except block is executed.