Sumit Rastogi Assignment

1.What is the role of try and exception block?

Ans1.) A try-except block, also known as an exception handling block, is a programming construct used to handle errors and exceptions in code. It allows a program to gracefully handle unexpected situations or errors that might occur during its execution, preventing the program from crashing and providing a way to recover from the error.

a.Error Handling: It allows you to handle specific error situations gracefully, providing meaningful error messages or alternative actions.

b. Program Stability: By catching exceptions, your program can continue running after encountering errors, which helps prevent crashes and ensures a more stable user experience.

c. Debugging: Catching exceptions allows you to obtain information about errors, which is useful for debugging and improving the overall quality of your code.

d. User-Friendly: You can present user-friendly error messages rather than exposing technical details to users.

e. Fallbacks: You can define alternative actions or fallback strategies in case of errors.

2.What is the syntax for a basic try-except block?

Ans2.) Python

```
    # Code that might raise an exception
    # ...
except SomeExceptionType:
 # Code to handle the exception
  # ...
 exampletry:
num = 10 / 0 # This will raise a ZeroDivisionError
except ZeroDivisionError:
 print ("Error: Division by zero!")
```

2.What happens if an exception occurs inside a try block and there is no matching except block?

Ans3.) If an exception occurs inside a try block and there is no matching except block to catch that specific type of exception, the exception will propagate up the call stack until a matching except block is found or until the program terminates.

Here's what typically happens when there's no matching except block for a raised exception:

a. Program Termination: If the exception propagates all the way up the call stack to the top-level code and is not caught by any except block, the program will terminate abruptly.

b. Error Information: The error message might contain information about the type of exception, the line number or location in the code where the exception occurred, and possibly a stack trace that shows the sequence of function calls leading up to the exception.

To handle exceptions more effectively:

c. Identify the specific types of exceptions that can occur in your code.

d. Add appropriate except blocks to handle those specific exception types.

e. Optionally, include a more general except block (without specifying an exception type) at the end to catch unexpected exceptions and handle them gracefully, such as by providing a user-friendly error message and logging the details for debugging.

4. What is the difference between using a bare except block and specifying a specific exception type?

Ans4.) Specifying a Specific Exception Type:

a.) Focused Handling: You can provide specific error handling logic for different types of exceptions.

b.) Clarity: It's clear which exceptions you are prepared to handle.

c.) Avoid Overgeneralization: You avoid unintentionally catching and suppressing exceptions that you might not be able to handle appropriately.

Bare except Block (Catch-All):

a.) Generic Handling: You can provide a common fallback for all exceptions.

b.) Last Resort: It can serve as a last-resort mechanism to prevent abrupt program termination due to unhandled exceptions.

5.Can you have nested try-except blocks in Python? If yes, then give an example.

Ans5.) Yes, you can have nested try-except blocks in Python. This means you can place a try-except block inside another try block or inside an except block.

```
try:
    # Outer try block
    num = int (input ("Enter a number: "))

    try:
        # Inner try block
        result = 10 / num
        print ("Result:", result)
    except Zero Division Error:
        print ("Error: Division by zero inside the inner try block.")
    except Value Error:
        print ("Error: Invalid input inside the inner try block.")
except Value Error:
    print ("Error: Invalid input inside the outer try block.")
```

In this example:

a. The outer try-except block captures potential Value Error exceptions when trying to convert the user input to an integer.

b. If the input is valid, the inner try-except block calculates the result of dividing 10 by the user-provided number. It handles Zero Division Error and Value Error exceptions separately inside this block.

6.Can we use multiple exception blocks, if yes then give an example.

Ans6. Yes, you can use multiple exception blocks (also known as multiple except blocks) in languages like Python to catch and handle different types of exceptions separately.

```
try:
    num = int (input ("Enter a number: "))
    result = 10 / num
    print ("Result:", result)
except Value Error:
    print ("Error: Invalid input. Please enter a valid number.")
```

```
except Zero Division Error:
    print ("Error: Division by zero is not allowed.")
except Exception as e:
    print (f" An unexpected error occurred: {e}")
```

In this example:
The first except block catches a Value Error that might occur if the user enters something that cannot be converted to an integer.
The second except block catches a Zero Division Error that might occur if the user enters zero as the input.
The third except block is a generic exception handler that catches any other unexpected exceptions. It prints a message indicating that an unexpected error occurred, along with the details of the exception.

7.Write the reason due to which following errors are raised:

- EOF Error
- Floating Point Error
- Index Error
- MemoryError
- Overflow Error
- Tab Error
- Value Error

Ans7.) a. EOF Error:
Reason: EOF (End of File) Error occurs when an operation that requires data from a file is attempted, but the end of the file is reached before the operation completes. This error indicates that the program attempted to read beyond the end of a file or input stream.

b. Floating Point Error:
Reason: A Floating Point Error, commonly known as a Floating Point Exception, occurs during arithmetic operations involving floating-point numbers (numbers with decimal points) when the result is undefined or can't be represented accurately due to limitations in the floating-point representation.

c. Index Error:
Reason: An Index Error (or Index Error) occurs when attempting to access an element of a sequence (such as a list or tuple) using an index that is outside the valid range of indices for that sequence. This typically happens when the index is negative, greater than or equal to the length of the sequence, or when working with empty sequences.

d. Memory Error:
Reason: A Memory Error occurs when a program tries to allocate more memory (RAM) than the system can provide. This usually happens when a program requests more memory than is available due to resource limitations.

e. Overflow Error:
Reason: An Overflow Error occurs in numeric calculations when the result of an arithmetic operation exceeds the range of values that can be represented by the data type being used. For example, integer overflow can occur if the result of an addition or multiplication operation exceeds the maximum value that can be stored in the data type.

g. Value Error:
Reason: A Value Error occurs when a function or operation receives an argument of the correct data type, but the argument's value is inappropriate or outside the expected range.

8. Write code for the following given scenario and add try-exception block to it.
    a. Program to divide two numbers
    b. Program to convert a string to an integer
    c. Program to access an element in a list
    d. Program to handle a specific exception
    e. Program to handle any exception

ANS8.) Certainly, here are code examples for each of the scenarios you've mentioned, along with the corresponding try-except blocks:

a. Program to Divide Two Numbers:

```
try:
    dividend = int (input ("Enter the dividend: "))
    divisor = int (input ("Enter the divisor: "))
    result = dividend / divisor
    print ("Result:", result)
except Zero Division Error:
    print ("Error: Division by zero is not allowed.")
except Value Error:
    print ("Error: Please enter valid numeric inputs.")
except Exception as e:
    print (f" An unexpected error occurred: {e}")
```

b. Program to Convert a String to an Integer:

```
try:
    num_str = input ("Enter a number: ")
    num = int(num_str)
    print ("Converted integer:", num)
except Value Error:
    print ("Error: Invalid input. Please enter a valid number.")
except Exception as e:
    print (f" An unexpected error occurred: {e}")
```

c. Program to Access an Element in a List:

```
try:
    my_list = [1, 2, 3, 4, 5]
    index = int (input ("Enter an index: "))
    value = my_list [index]
    print ("Value at index:", value)
except Index Error:
    print ("Error: Index out of range.")
except Value Error:
    print ("Error: Invalid input. Please enter a valid index.")
except Exception as e:
    print (f" An unexpected error occurred: {e}")
```

d. Program to Handle a Specific Exception:

```
try:
    num = int (input ("Enter a number: "))
    if num < 0:
        raise Value Error ("Negative numbers are not allowed.")
    print ("Number:", num)
except Value Error as ve:
    print (f" Value Error: {ve}")
except Exception as e:
    print (f" An unexpected error occurred: {e}")
```

e. Program to Handle Any Exception:

```
try:
    # Your code here
```

```
except Exception as e:
    print (f" An exception occurred: {e}")
```