1. What is the role of the 'else' block in a try-except statement? Provide an example scenario where it would be useful.

Ans – in python the else block in a try except statement is executed only if no exceptions are raised in the try block, for an example you're opening a file. If the file is found, you want to read and print its contents. If it’s not found, show an error message.

If the file **doesn’t exist**, Python runs the except block.

If the file **opens successfully**, Python runs the else block to read and print the content

1. Can a try-except block be nested inside another try-except block? Explain with an example.

Ans – Yes in python , your can nest a try except block inside another try except block, this is called a nested try exce[t and it is used when you want to handle different errors separately at different levels of your code.

Example - try:

num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

try:

result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

print("Inner Error: Cannot divide by zero.")

except ValueError:

print("Outer Error: Please enter valid numbers.")

in try method if both inputs are valid , this block tried to divide them, if num2 = 0 , it raises a zero Division Error , this will be caught by the inner except Zero division error

we can handle nest try except block to handle different types of error at different level, inner block handles specific issues like division error , outer block handles broader issues like input validation

1. How can you create a custom exception class in Python? Provide an example that demonstrates its usage.

* Ans – The class UnderageError inherits from Exception.
* When age is less than 18, the program raises this custom exception with a meaningful message.
* The try-except block catches the custom exception and prints the message.
* If the input is not a number, it catches a ValueError.

1. What are some common exceptions that are built-in to Python?

Ans – ValueError , TypeError , ZeroDivisionError, indexError, KeyError, FilenotfoundError,Attribute ERROR , import Error, indentation Error ,

1. What is logging in Python, and why is it important in software development?

Ans – logging in python is the process of recording messages or information about a program execution, these messages can include information about the program flow, warning, errors, or debugging details. Python provides a built-in module called logging that helps developers create logs easily,

Why login important in software development? – for debugging, monitoring, auditing, maintenance, error tracking,

1. Explain the purpose of log levels in Python logging and provide examples of when each log level would be appropriate.

Ans –

1. What are log formatters in Python logging, and how can you customise the log message format using formatters?

Ans – log levels are use to categorize the importance and severity of messages generated by a program during execution, they help developers and system admisnitrators understand , what type of events are happeninig from detailed debugging info to critical failures , we can control which msgs t orecord or display , making it easier to focus on the most relevant information,

1. How can you set up logging to capture log messages from multiple modules or classes in a Python application?

Ans – to capture log messages from multiple modules or classes in a python application , you should set up a centralized configuration, module or class should create its own logger using , logging.get.logger. this way ,all loggers follow the same configuration.

1. What is the difference between the logging and print statements in Python? When should you use logging over print statements in a real-world application?

Ans – logging is a flexible and powerful way to record messages with different severity levels , like debug , info , warning , error , loggin can be configured to save messages to files .

Print – simply display output ot the console and are mainly used for quick debugging or showing simple messages during development, thet do not have levels of importance

WHEN TO USE LOGGING OVER PRINT: logging provide better control filtering and persistence of messages , , when u want to keep a record of program events or errors that can be reviewed later, we use print for print for termperorary debugging during development ,

1. Create a Python program that logs an error message to the console and a file named "errors.log" if an exception occurs during the program's execution. The error message should include the exception type and a timestamp.

Ans - import logging

import datetime

# Configure logging to file

logging.basicConfig(

filename='errors.log',

level=logging.ERROR,

format='%(asctime)s - %(levelname)s - %(message)s',

filemode='a' # Append mode

)

try:

# Example code that may raise an exception

x = 10 / 0 # Division by zero will raise an exception

except Exception as e:

# Get current time and exception type

error\_type = type(e).\_\_name\_\_

timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# Create error message

error\_message = f"{timestamp} - Error occurred: {error\_type} - {str(e)}"

# Log to file

logging.error(error\_message)

# Print to console

print(error\_message)