MODULE – 5 ASSIGNMENT

File Handling & Exception Handling

1)Write the code in python to open a file named “try.txt”?

```
file = open("try.txt", "r")

content = file.read()

print(content)

file.close()
```

2)What is the purpose of ‘r’ as a prefix in the given statement?

```
f = open(r, “d:\color\flower.txt”)
```

The r prefix in the statement f = open(r, "d:\color\flower.txt") indicates a raw string literal in Python.

Here's what the r prefix does:

When you prefix a string with r, Python treats the backslashes (\) as literal characters and does not interpret them as escape sequences.

Without the r prefix, Python would interpret certain character combinations starting with \ as special characters. For example, \n is interpreted as a newline character, \t as a tab character, and so on.

In the given statement:

python

```
f = open(r"d:\color\flower.txt")
```

The r prefix ensures that the backslashes in the file path are treated as literal backslashes and not as escape characters. This means the string r"d:\color\flower.txt" is interpreted exactly as d:\color\flower.txt.

Without the r prefix, you would need to escape each backslash by doubling it:

f = open("d:\\color\\flower.txt")

This is because, without the raw string prefix, Python would interpret \c and \f as escape sequences, which are not valid and would result in an error or unexpected behavior.

So, the purpose of r in this context is to simplify the writing of file paths by avoiding the need to escape backslashes.

3)Write a note on the following

A. Purpose of Exception Handling
B. Try block
C. Except block
D. Else block
E. Finally block
F. Built-in exceptions

A. Purpose of Exception Handling

Exception handling is a programming construct used to manage errors and exceptional conditions in a program gracefully. The primary purposes of exception handling are:

Graceful Degradation: Allows a program to continue running even after encountering an error, providing a way to recover from exceptions.

Resource Management: Ensures that resources such as file handles, database connections, and network connections are properly closed or released, even in the event of an error.

Separation of Error-Handling Code: Separates the error-handling code from the main logic of the program, improving readability and maintainability.

Debugging Aid: Provides meaningful error messages and stack traces that help developers identify and fix issues.

Program Robustness: Enhances the robustness and reliability of the program by handling unexpected conditions gracefully.

B. Try Block

A try block is used to wrap the code that might generate an exception. It defines a block of code in which exceptions are anticipated and provides a mechanism to catch and handle those exceptions. The syntax is:

```
try:
    # Code that might raise an exception
    risky_operation()
```

C. Except Block

An except block is used to handle the exceptions that occur in the associated try block. It catches specific exceptions and provides a way to respond to them. Multiple except blocks can be used to handle different types of exceptions. The syntax is:

```
try:
    # Code that might raise an exception
    risky_operation()
except SpecificException as e:
    # Code to handle the exception
    handle_exception(e)
```

D. Else Block

An else block follows all except blocks and runs if no exceptions were raised in the try block. It is useful for code that should only run if the try block was successful. The syntax is:

```
try:
    # Code that might raise an exception
    risky_operation()
except SpecificException as e:
```

```
    # Code to handle the exception
    handle_exception(e)
else:
    # Code to run if no exception occurs
    operation_successful()
```

E. Finally Block

A finally block is used to define cleanup actions that must be executed under all circumstances, regardless of whether an exception was raised or not. This block runs after the try and except blocks, making it ideal for resource management tasks like closing files or releasing locks. The syntax is:

```
try:
    # Code that might raise an exception
    risky_operation()
except SpecificException as e:
    # Code to handle the exception
    handle_exception(e)
finally:
    # Code to run regardless of what happened before
    cleanup_operations()
```

F. Built-in Exceptions

Python provides a range of built-in exceptions that cover various error conditions. Some common built-in exceptions include:

Exception: The base class for all exceptions.

Arithmetic Error: Base class for arithmetic-related errors, including:

ZeroDivisionError: Raised when division or modulo by zero occurs.

OverflowError: Raised when the result of an arithmetic operation is too large to be expressed within the range of the numeric type.

FloatingPointError: Raised when a floating-point operation fails.

AttributeError: Raised when an attribute reference or assignment fails.

IndexError: Raised when a sequence subscript is out of range.

KeyError: Raised when a dictionary key is not found.

NameError: Raised when a local or global name is not found.

TypeError: Raised when an operation or function is applied to an object of inappropriate type.

ValueError: Raised when a function receives an argument of the correct type but inappropriate value.

FileNotFoundError: Raised when a file or directory is requested but does not exist.

IOError: Raised when an input/output operation fails.

ImportError: Raised when an import statement fails to find the module definition or when a from ... import fails to find a name that is to be imported.

These exceptions help in identifying and handling specific error conditions effectively in a Python program.

4) Write 2 Custom exceptions?

Creating custom exceptions in Python involves defining new exception classes that inherit from the built-in Exception class or one of its subclasses. Here are two examples of custom exceptions:

Example 1: InvalidAgeError

This custom exception can be used to handle cases where an invalid age is provided.

Example 2: InsufficientFundsError

This custom exception can be used to handle cases where a withdrawal amount exceeds the available balance in a bank account.

In these examples:

InvalidAgeError is raised when an invalid age is provided.

InsufficientFundsError is raised when a withdrawal amount exceeds the available balance.

Both custom exceptions inherit from the built-in Exception class and provide additional context and custom error messages.