

**K. J. Somaiya College of Engineering, Mumbai-77**

**Batch: C2-2                      Roll No.: 16010124107**  
**Experiment / assignment / tutorial No.**  
**Grade: AA / AB / BB / BC / CC / CD / DD**  
**Signature of the Staff In-charge with date**

**TITLE:** Write a program to demonstrate exception handling and file handling mechanism in Python

**AIM:** 1) Write a program to demonstrate exception handling mechanism in Python  
2) Write a program to demonstrate File handling mechanism in Python

**OUTCOME:** Students will be able to

**CO1:** Formulate a problem statement and develop the logic (algorithm/flowchart) for its solution.

**CO5:** Apply the concept of exception handling and file handling in Python.

To demonstrate how exceptions can be used to improve our programs' robustness, reliability, and maintainability by handling unexpected errors effectively, also demonstrate the File Handling mechanism in Python.

---

**Resource Needed: Python IDE**

---

**Books/ Journals/ Websites referred:**

1. Reema Thareja, *Python Programming: Using Problem-Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018, India
3. <https://www.geeksforgeeks.org/python-strings/?ref=lbp>

---

**Theory:**

**Exception Handling in Python:**

**What is Exception?**

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

**Department of Department of Science and Humanities**

## K. J. Somaiya College of Engineering, Mumbai-77

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

### Built-in Exceptions

Commonly occurring exceptions are usually defined in the compiler/interpreter. These are called built-in exceptions. Python's standard library is an extensive collection of built-in exceptions that deal with commonly occurring errors (exceptions) by providing standardized solutions for such errors. When any built-in exception occurs, the appropriate exception handler code displays the reason along with the raised exception name. The programmer then has to take appropriate action to handle it. Some of the commonly occurring built-in exceptions that can be raised in Python are explained below

Following is the list of common exception classes found in Python:

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax <code>obj.foo</code> , if <code>obj</code> has no member named <code>foo</code>
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by <code>next(iterator)</code> if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., <code>sqrt(-5)</code> )
ZeroDivisionError	Raised when any division operator used with 0 as divisor

### Handling an exception

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the **try:** block, include an **except:** statement, followed by a block of code that handles the problem as elegantly as possible.

#### The try-except else block:

##### Syntax:

```
try:
    You do your operations here;
except ExceptionI:
    If there is ExceptionI, then execute this block
else:
    If there is no Exception, then execute this block
```

## K. J. Somaiya College of Engineering, Mumbai-77

Here are few important points about the above-mentioned syntax:

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

### Example:

This example opens a file, writes content in the, file and comes out gracefully because there is no problem at all

```
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
except IOError:
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
    fh.close()
```

This produces the following result:

Written content in the file successfully

### Example:

This example tries to open a file where you do not have write permission, so it raises an exception

```
try:
    fh = open("testfile", "r")
    fh.write("This is my test file for exception handling!!")
except IOError:
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
```

This produces the following result:

Error: can't find file or read data

### The Except Clause with No Exceptions:

You can also use the except statement with no exceptions defined as follows:

```
try:
    You do your operations here;
    .....
```

## K. J. Somaiya College of Engineering, Mumbai-77

```
except:
    If there is any exception, then execute this block
    .....
else
    If there is no exception, then execute this block
    .....
```

This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

### The Except Clause with Multiple Exceptions:

You can also use the same except statement to handle multiple exceptions as follows:

```
try:
    You do your operations here;
    .....
except(Exception1[, Exception2[,...ExceptionN]]):
    If there is any exception from the given exception list,
    then execute this block.
    .....
else:
    If there is no exception then execute this block.
```

### The try-finally Clause:

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this

```
try:
    You do your operations here;
    .....
    Due to any exception, this may be skipped.
finally:
    This would always be executed.
    .....
```

You cannot use else clause as well along with a finally clause.

### Example

```
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
finally:
    print "Error: can't find file or read data"
```

### K. J. Somaiya College of Engineering, Mumbai-77

If you do not have permission to open the file in writing mode, then this will produce the following result

Error: can't find file or read data

The same example can be written more cleanly as follows:

```
try:
    fh = open("testfile", "w")
    try:
        fh.write("This is my test file for exception handling!!")
    finally:
        print "Going to close the file"
        fh.close()
except IOError:
    print "Error: can't find file or read data"
```

When an exception is thrown in the try block, the execution immediately passes to the finally block. After all the statements in the finally block are executed, the exception is raised again and is handled in the except statements if present in the next higher layer of the try-except statement.

#### The argument of an Exception:

An exception can have an argument, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause as follows –

```
try:
    You do your operations here;
    .....
except ExceptionType, Argument:
    You can print value of Argument here...
```

If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.

This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.

#### Example

Following is an example for a single exception

```
# Define a function here.
def temp_convert(var):
    try:
```

### K. J. Somaiya College of Engineering, Mumbai-77

```
return int(var)
except ValueError, Argument:
    print "The argument does not contain numbers\n", Argument

# Call above function here.
temp_convert("xyz");
```

This produces the following result:

The argument does not contain numbers  
invalid literal for int() with base 10: 'xyz'

### Raising an Exceptions:

You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise** statement is as follows.

Syntax:

```
raise [Exception [, args [, traceback]]]
```

Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

### Example

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows:

```
def functionName( level ):
    if level < 1:
        raise "Invalid level!", level
        # The code below to this would not be executed
        # if we raise the exception
```

**Note:** In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string. For example, to capture above exception, we must write the except clause as follows –

```
try:
    Business Logic here...
except "Invalid level!":
    Exception handling here...
else:
    Rest of the code here...
```

## **K. J. Somaiya College of Engineering, Mumbai-77**

### **User-Defined Exceptions**

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

```
class Networkerror(RuntimeError):  
    def __init__(self, arg):  
        self.args = arg
```

So once you defined above class, you can raise the exception as follows –

```
try:  
    raise Networkerror("Bad hostname")  
except Networkerror e:  
    print e.args
```

### **File Handling in Python**

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

The file handling plays an important role when the data needs to be stored permanently in the file. A file is a named location on a disk to store related information. We can access the stored information (non-volatile) after the program termination.

The file-handling implementation is slightly lengthy or complicated in the other programming languages, but it is easier and shorter in Python.

#### **File operation can be done in the following order:**

Open a file  
Read or write - Performing operation  
Close the file

#### **Opening a file:**

### K. J. Somaiya College of Engineering, Mumbai-77

Python provides an open() function that accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

**Syntax:**

file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	Access mode	Description
1	r	It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.



### K. J. Somaiya College of Engineering, Mumbai-77

10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

#### Example:

```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")
if fileptr:
    print("file is opened successfully")
```

#### Output:

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

In the above code, we have passed the filename as a first argument and opened file in read mode as we mentioned r as the second argument. The fileptr holds the file object and if the file is opened successfully, it will execute the print statement.

#### The close() method:

Once all the operations are done on the file, we must close it through our Python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally using the file system which is the currently opened in Python; hence it is good practice to close the file once all the operations are done.

The syntax to use the close() method is given below.

#### Syntax:

```
fileobject.close()
```

#### Example:

```
try:
    fileptr = open("file.txt")
    # perform file operations
finally:
    fileptr.close()
```

### Reading and Writing Files

Department of Department of Science and Humanities

### **K. J. Somaiya College of Engineering, Mumbai-77**

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

#### **The write() Method**

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string

#### **Syntax:**

```
fileObject.write(string)
```

#### **Example:**

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opened file
fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

#### **Output:**

```
Python is a great language.
Yeah its great!!
```

#### **The read() Method**

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

#### **Syntax:**

```
fileObject.read([count])
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

#### **Example:**

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opened file
fo.close()
```

#### **Output**

```
Read String is : Python is
```

## K. J. Somaiya College of Engineering, Mumbai-77

### File Positions

The tell() method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The seek(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.

If from is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

### Example:

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10)
print "Read String is : ", str

# Check current position
position = fo.tell()
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10)
print "Again read String is : ", str
# Close opened file
fo.close()
```

### Renaming and Deleting Files

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

### The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

### Syntax:

```
os.rename(current_file_name, new_file_name)
```

### Example:

Following is the example to rename an existing file test1.txt –

```
import os
```

```
# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

## K. J. Somaiya College of Engineering, Mumbai-77

### The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

### Syntax:

```
os.remove(file_name)
```

### Example:

Following is the example of deleting an existing file test2.txt

```
—  
import os  
  
# Delete file test2.txt  
os.remove("text2.txt")
```

### Problem Definition:

1. Write a program in which we prompt users to enter personal details like name and surname, which should be strings, age should be an integer, and height and weight should be float. Whenever the user enters input of the incorrect type, keep prompting the user for the same value until it is entered correctly. Give the user sensible feedback
2. Write a program to create a file employeeedetails.txt" which stores the Employee details by adding their Employee Id, Name, and Department into it using the following format:

EmpId	Name	Department
1601001	Abc	Computer
1601003	Xyz	IT

Obtain the details for EmpId from the user.

### Books/ Journals/ Websites referred:

1. Reema Thareja, Python Programming: Using Problem-Solving Approach, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, Python Programming: A modular Approach, Pearson India, Second Edition 2018, India

## K. J. Somaiya College of Engineering, Mumbai-77

### Implementation details:

Problem 1:

Code:

```

J unit.java 1  ashwerahasan.py X
ashwerahasan.py > ...
1  try:
2      name = input("Enter your first name: ")
3  except ValueError:
4      print("Name should be a string.")
5  try:
6      surname = input("Enter your surname: ")
7  except ValueError:
8      print("Name should be a string.")
9  try:
10     age = int(input("Enter your age: "))
11 except ValueError:
12     print("Age should be a number. ")
13 try:
14     height = float(input("Enter your height: "))
15     weight = float(input("Enter your weight: "))
16 except ValueError:
17     print("Height and weight can be only floating point numbers. ")

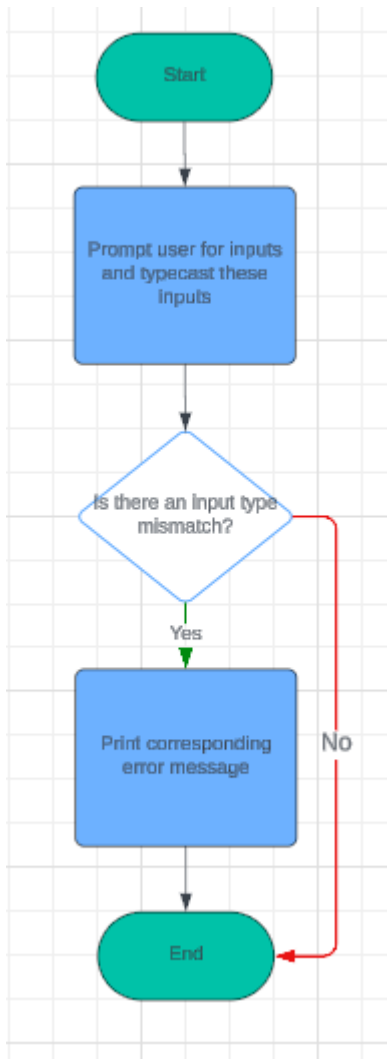
```

Algorithm:

1. Start
2. Prompt user for inputs and typecast these inputs:
  - Prompt for first name as a string.
  - Prompt for surname as a string.
  - Prompt for age and typecast to integer.
  - Prompt for height and weight and typecast to float.
3. If input type mismatch occurs, print corresponding error message:
  - "Name should be a string."
  - "Age should be a number."
  - "Height and weight can be only floating point numbers."
4. End

Flowchart:

**K. J. Somaiya College of Engineering, Mumbai-77**



Problem 2:  
Code:

## K. J. Somaiya College of Engineering, Mumbai-77

```

Go Run ... Python
ashwerahasan.py X
ashwerahasan.py > ...
1 file = open("employeedetails.txt","w+")
2 file.write("EmpId\tName\tDepartment\n")
3 number = int(input("Enter number of employees: "))
4 while(number>0):
5     empid = int(input("Enter employee ID: "))
6     empname = input("Enter employee name: ")
7     dept = input("Enter employee department: ")
8     file.write(f"{empid}\t{empname}\t{dept}\n")
9     number-=1
10 file.seek(0) # to read from the beginning of the file
11 content = file.read()
12 print(content)
13 file.close()

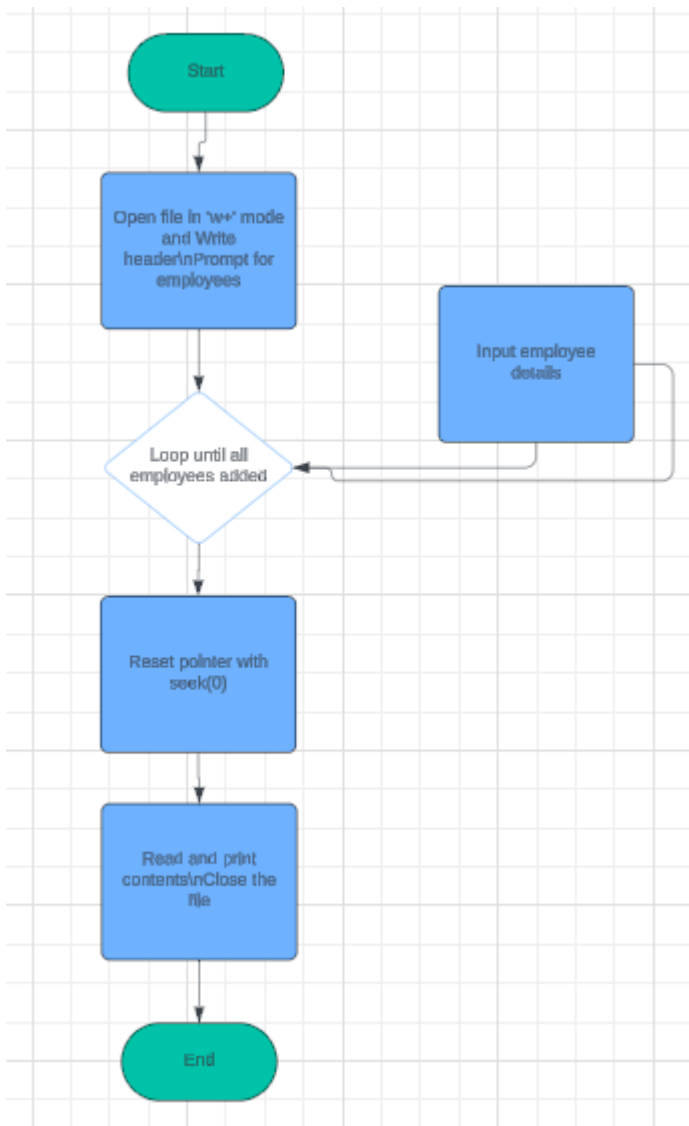
```

Algorithm:

1. Start
2. Open the file in write and read mode ("w+").
3. Write the header line ("EmpId\tName\tDepartment\n").
4. Prompt for the number of employees.
5. Loop until the specified number of employees is reached:
  - a. Input employee ID.
  - b. Input employee name.
  - c. Input employee department.
  - d. Write the employee details to the file in the format: "{empid}\t{empname}\t{dept}\n".
6. Reset the file pointer to the beginning using file.seek(0).
7. Read the contents of the file.
8. Print the contents.
9. Close the file.
10. Stop

Flowchart:

**K. J. Somaiya College of Engineering, Mumbai-77**



**Output(s):**

```
bs\debugpy\adapter/../../debugpy\launcher'
Enter your first name: Ashwera
Enter your surname: Hasan
Enter your age: 90.5
Age should be a number.
Enter your height: 5.1
Enter your weight: 30
PS C:\Users\syeda\OneDrive\Desktop\Python>
```



**K. J. Somaiya College of Engineering, Mumbai-77**

```
Enter number of employees: 2
Enter employee ID: 160102
Enter employee name: Ashwera
Enter employee department: Computer
Enter employee ID: 104234
Enter employee name: Danish
Enter employee department: Finance
EmpId    Name    Department
160102   Ashwera  Computer
104234   Danish   Finance
```

**Conclusion:**

We can handle errors in Python using try-except blocks for quick input validation. The input() function enables us to gather user input for various data types. Using the except keyword, we can continue the code when an error is encountered and tell the user about it effectively.

We can also create files using the open() function with w+ mode to store data. Then, we write to the file for as many times as the user prompts. Using file.seek(0), we return the pointer to the beginning of the file, and then read the content to print it. Overall, these skills enhance our ability to manage user input and data storage effectively in Python.

**Post Lab Questions:**

1. Write a program that takes a list of numbers as input from the user and calculates their average. If the list is empty, raise a custom exception EmptyListError with an appropriate error message.

## K. J. Somaiya College of Engineering, Mumbai-77

```

Go Run ... ← → Python
ashwerahasan.py X
ashwerahasan.py > ...
1 class EmptyListError(Exception):
2     pass
3
4 def calculate_average(numbers):
5     if not numbers:
6         raise EmptyListError("The list is empty. Cannot calculate the average.")
7     return sum(numbers) / len(numbers)
8
9 user_input = input("Enter numbers separated by spaces: ")
10 number_list = [int(num) for num in user_input.split()]
11
12 try:
13     average = calculate_average(number_list)
14     print(f"The average is: {average}")
15 except EmptyListError as e:
16     print(e)

```

```

Enter numbers separated by spaces: 1 2 3 4 5
The average is: 3.0
PS C:\Users\syeda\OneDrive\Desktop\Python>

```

2. What is the purpose of the "finally" block in a try-except-finally block?

The purpose of the finally block in a try-except-finally block is to define code that will always execute, regardless of whether an exception was raised or handled. This is useful for performing cleanup actions, such as closing files or releasing resources, ensuring that these actions occur even if an error occurs in the try block.

3. Virtual lab on File Operation: <https://python-iitk.vlabs.ac.in/exp/file-operators>

## K. J. Somaiya College of Engineering, Mumbai-77

The screenshot shows a web application titled "File Operation" with a red header "Analysis of File Operators". It contains three panels:

- Demonstration:** Contains the text "To Know About File Handling In Python." and buttons for "Start", "Next", and "Reset".
- Step Execute:** Shows the following Python code:
 

```
file=open('mitra.txt','w')
file.write('Hello! Developers How is your work going on ??')
file.close()
file=open('mitra.txt','r')
print(file.read())
file.close()
```
- Result:** Shows the execution steps and output:
 

```
file->mitra.txt(write mode)
mitra.txt->write: Hello! Developers How is your work going on ??
file->close
file->mitra.txt(read mode)
Hello! Developers How is your work goin on ??
file->close
```

4. Write a program that prompts the user for a file name and then reads and prints the contents of the requested file in the upper case.

The screenshot shows a Python IDE with the following code in `ashwerahasan.py`:

```
1 file_name = input("Enter the file name: ")
2
3 try:
4     with open(file_name, "r") as file:
5         content = file.read()
6         print(content.upper())
7 except FileNotFoundError:
8     print("File not found. Please check the file name.")
```

The terminal output shows the program execution:

```
Enter the file name: helloworld.txt
HELLO WORLD!
PS C:\Users\syeda\OneDrive\Desktop\Python>
```

**K. J. Somaiya College of Engineering, Mumbai-77**

**Date: 26/10/2024**

**Signature of faculty in-charge**

**Department of Department of Science and Humanities**

PP/I/July-November\_2024\_Page No.-\_\_\_\_\_