Tutorial_Session4

September 28, 2018

1 FILE HANDLING AND ERROR HANDLING

1.1 File Handling

- Files provide a means of communication between the program and the outside world.
- A file is a stream of bytes, comprising data of interest.
- **File Operations:** > * Create a file > * Reading from a file > * Writing to a file > * Append to a file > * **Rename a file** > * **Delete a file**

1.1.1 Opening a File

- Built-in function used: **open()**
- This function takes the name of the file as the first argument. The second argument indicates the mode for accessing the file.
- Modes for opening a file:
  - read(r): to read the file
  - write(w): to write to the file
  - append(a): to write at the end of the file.

Syntax:

```
f = open(file_name, access_mode)
```

- Opening a non-existent file in w or a mode creates a new file with the given name
- However, opening a non-existent file in r mode leads to an error.

```
In [1]: f = open('PYTHON','w')
```

1.1.2 Writing to a File

- Built-in function used: **write()**
- To use write function, specify name of the file object, followed by the dot operator (.), followed by the name of the function.
- Note that apart from writing the given data into a file, the function write also returns the number of characters written into the file.

```
In [2]: f.write('''Python:
        Python is Simple.
        Simple syntax.''')
        f.close()

In [3]: f = open('PYTHON','w')
        for i in range(10):
             f.write("This is line %d\n" % (i+1))
        f.close()
```

1.1.3 Reading a File

- Built-in function used: **read()**
- To use read function, specify name of the file object, followed by the dot operator (.), followed by the name of the function
- The read() function retrieves the contents of the entire file.

```
In [4]: f.read() #attempt to read file without read mode on.


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-4-99d52ada1abc> in <module>()
    ----> 1 f.read() #attempt to read file without read mode on.


        ValueError: I/O operation on closed file.


In [ ]: f = open('PYTHON','r')
        f.read()

In [ ]: f.read() # reading a file reached EOF yields empty string
```

We can read a fixed number of bytes from the file by specifying the number of bytes as the argument to read function.

```
In [ ]: f = open('PYTHON','r')
        f.close()

In [ ]: f.read(6)
```

Displaying the multi-line string using the print function

```
In [ ]: f = open('PYTHON','r')
        print(f.read())
```

Append Mode

```
In [ ]: f = open('PYTHON','a')
        f.write("\nPython is interactive1")

In [ ]: f.close()
```

Rename a file

```
In [ ]: import os
        os.rename('a.txt','g.txt')
```

Delete a file

```
In [ ]: import os
        if os.path.exists("g.txt"):
          os.remove("g.txt")
        else:
          print("The file does not exist")
```

Create a file

```
In [ ]: f = open("g.txt", "x")
```

1.1.4 Function close

- When a file is no longer required for reading or writing, it should be closed by invoking the function close

```
In [ ]: f.close()
```

The function close also saves a file, which was opened for writing. Once a file is closed, it cannot be read or written any further unless it is opened again and an attempt to access the file results in an I/O (input/output) error:

```
In [ ]: f.write('Python was developed by Guido Van Rossum in 1991.')
```

1.1.5 Functions readline and readlines

readline function: reads a stream of bytes beginning the current position until a newline character is encountered

```
In [ ]: f = open('PYTHON','r')
        line = f.readline()
        print('line1:', line)
        line = f.readline()
        print('line2:', line)
        line = f.readline()
        print('line3:', line)
        line = f.readline()
        print('line4:', line)
        f.close()
```

```
In [ ]: f=open('PYTHON','r')
        f.readline()

In [ ]: f.tell()
```

readlines function: returns all the remaining lines of the file in the form of a list

```
In [ ]: f = open('PYTHON','r')
        lines = f.readlines()
        print(lines)
        f.close()

In [ ]: f= open ('PYTHON','r')

        line=f.readline()

        while (line!=''):
            print(line)
            line=f.readline()
```

1.1.6 Function writelines

writelines function: takes a list of lines to be written in the file as an argument

```
In [ ]: f = open('PYTHON', 'w')
        description = ['Python:\n', 'Python is simple.\n']
        f.writelines(description)
        f.close()

        f = open('PYTHON', 'r')
        print(f.read())
        f.close()
```

1.1.7 Functions seek and tell

- **seek()**: to reach desired position in a file Syntax:

  seek(offset) # offset indicates the number of bytes to be moved # Returns the new absolute position

- **tell()**: to find current position of the file object

```
In [ ]: f = open('PYTHON','r')
        print(f.readline())
        f.seek(0) # Changes the current file position to the beginning of the file
        print('After seeking file pointer in the beginning\n')
        print(f.readline())

In [ ]: f.seek(0)
        f.read(4)
```

```
In [ ]: f.tell()
```

Writing Structures to file: Pickle > Pickling refers to the process of converting the structure to a byte stream before writing to the fle.

```
> While reading the contents of the fle, a reverse process called unpickling is used to convert

> We can save the memory and improve the performance by directlt writing the binary data on the

> It can't be opened by any other applications
```

```
In [ ]: import pickle
        def main():
            '''
            Objective: To write and read a list and dictionary to and from a file
            Input Parameter: None
            Return Value: None
            '''
            f = open('file1', 'wb')
            pickle.dump(['hello', 'world'], f)
            pickle.dump({1:'one', 2:'two'}, f)
            f.close()

            f = open('file1', 'rb')
            value1 = pickle.load(f)
            value2 = pickle.load(f)
            print(value1, value2)
            f.close()

        if __name__ == '__main__':
            main()
```

1.2 Error Handling

- Error occurs when something goes wrong.
- The errors in Python programming may be categorized as:
  - Syntax Errors
  - Exceptions

1.3 Syntax Error

A syntax error occurs when a rule of Python grammar is violated.

```
In [ ]: print('Hello)   # missing quote mark at the end of the string
```

```
In [ ]: for i in range(0, 10)   # missing colon in the for statement
```

1.4 Exceptions

- Errors that occur at execution time.

- These errors disrupt the flow of the program at a run-time by terminating the execution at the point of occurrence of the error.
- We have noticed that whenever an exception occurs, a Traceback object is displayed which includes error name, its description, and the point of occurrence of the error such as line number.

1.4.1 NameError

This exception occurs whenever a name that appears in a statement is not found globally.

```
In [ ]: marks = Input('Enter your marks')
```

```
In [ ]: price=10
        print(Price)
```

1.4.2 TypeError

This exception occurs when an operation or function is applied to an object of inappropriate type.

```
In [ ]: 'sum of 2 and 3 is ' + 5
```

1.4.3 ValueError

This exception occurs whenever an inappropriate argument value, even though of correct type, is used in a function call.

```
In [ ]: int('12345')
```

1.4.4 ZeroDivisionError

This exception occurs when we try to perform numeric division in which the denominator happens to be zero.

```
In [ ]: 78/(2+3-5)
```

1.4.5 FileNotFoundError

This exception occurs whenever there is an error related to input/output.

```
In [ ]: # opening a non-existent file for reading or reading a file opened in write mode
        f = open('passwordFile.txt')
```

1.4.6 IndexError

This exception occurs whenever we try to access an index that is out of a valid range.

```
In [ ]: colors = ['red', 'green', 'blue']
```

```
In [ ]: colors[4]
```

1.5 Handling Exceptions using Try...Except

- To prevent a program from terminating abruptly when an exception is raised, we need to handle it by catching it and taking appropriate action using the try…except clause.
  - try block: statements having potential to raise an exception.
  - except block: action to be performed when exception is raised.
  - finally block: executed irrespective of whether an exception is raised.

```
In [ ]: def main():
            '''
            Objective: To open a file for reading
            Input Parameter: None
            Return Value: None
            '''
            try:
                f = open('abc' , 'r')
            except IOError:
                print('Problem with Input Output... ')
            print('Program continues smoothly beyond try...except block')

        if __name__ == '__main__':
            main()
```

1.6 Accessing Description of Traceback object

- the name of the exception such as FileNotFoundError is followed by the keyword as, which is followed by a user-defined name such as err in the except clause.

```
In [ ]: import sys
        def main():
            '''
            Objective: To open a file for reading
            Input Parameter: None
            Return Value: None
            '''
            try:
                f = open('Temporary_File', 'r')
            except FileNotFoundError as desc:
                print('Problem with Input Output...\n', desc)
            print('Program continues smoothly beyond try...except block')

        if __name__ == '__main__':
            main()
```

1.7 Catch all possible exceptions

1. Enumerate all possible exceptions in the except clause in the form of a list.
2. Define an except clause for handling each exception separately.

3. Go for a defensive approach by placing the targeted group of statements in try block and specify an empty except clause.

```
In [ ]: import sys
        def main():
            '''
            Objective: To compute price per unit weight of an item
            Input Parameter: None
            Return Value: None
            '''
            price = input('Enter price of item purchased: ')
            weight = input('Enter weight of item purchased: ')
            try:
                if price == '': price = None
                price = float(price)
                if weight == '': weight = None
                weight = float(weight)
                assert price >= 0 and weight >= 0
                result = price / weight
            except (ValueError, TypeError, ZeroDivisionError):
                print('Invalid input provided by user \n' + \
                        str(sys.exc_info()))
            else:
                print('Price per unit weight: ', result)

        if __name__ == '__main__':
            main()
```

```
In [ ]: import sys
        def main():
            '''
            Objective: To compute price per unit weight of an item
            Input Parameter: None
            Return Value: None
            '''
            price = input('Enter price of item purchased: ')
            weight = input('Enter weight of item purchased: ')
            try:
                if price == '': price = None
                price = float(price)
                if weight == '': weight = None
                weight = float(weight)
                assert price >= 0 and weight >= 0
                result = price / weight
            except ValueError:
                print('Invalid inputs: ValueError')
            except TypeError:
                print('Invalid inputs: TypeError')
```

```
        except ZeroDivisionError:
            print('Invalid inputs: ZeroDivisionError')
        except:
            print('workshop')
        else:
            print('Price per unit weight:', result)

        print("Program terminates here!")

    if __name__ == '__main__':
        main()
```

1.8 If the script does not include the corresponding handler

- search for a handler is continued in the outer try…except block, and so on.

```
In [5]: import sys
        def main():
            '''
            Objective: To compute price per unit weight of an item
            Input Parameter: None
            Return Value: None
            '''
            price = input('Enter price of item purchased: ')
            weight = input('Enter weight of item purchased: ')
            try:
                if price == '': price = None
                try:
                    price = float(price)
                except ValueError:
                    print('Invalid price input: ValueError')
                if weight == '': weight = None
                try:
                    weight = float(weight)
                except ValueError:
                    print('Invalid weight input: ValueError')
                assert price >= 0 and weight >= 0
                result = price / weight
            except TypeError:
                print('Invalid inputs: TypeError')
            except ZeroDivisionError:
                print('Invalid inputs: ZeroDivisionError')
            #except:
             #   print(str(sys.exc_info()))
            else:
                print('Price per unit weight: ', result)

        if __name__ == '__main__':
```

```
            main()

Enter price of item purchased: -90
Enter weight of item purchased: 89



        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-5-044ff171591c> in <module>()
         31
         32 if __name__ == '__main__':
    ---> 33     main()


        <ipython-input-5-044ff171591c> in main()
         19         except ValueError:
         20             print('Invalid weight input: ValueError')
    ---> 21         assert price >= 0 and weight >= 0
         22         result = price / weight
         23     except TypeError:


        AssertionError:
```

1.9 Finally Clause

- If we wish to execute some action(s) irrespective of whether an exception is raised, such action(s) may be specifed in the finally clause.
- An exception may be raised explicitly using the keyword raise, followed by the name of the exception, followed by the string (enclosed in parentheses) to be displayed when the exception is raised.

```
In [6]: def main():
            '''
            Objective: To illustrate the use of raise and finally clauses
            Input Parameter: None
            Return Value: None
            '''
            marks = 110
            try:
                if marks < 0 or marks > 100:
                    raise ValueError('Marks out of range')
            finally:
                print('Bye')
            print('Program continues after handling exception')
```

```
        if __name__ == '__main__':
            main()
```

```
Bye
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-6-ebb7c519b664> in <module>()
         14
         15 if __name__ == '__main__':
    ---> 16     main()


        <ipython-input-6-ebb7c519b664> in main()
          8     try:
          9         if marks < 0 or marks > 100:
    ---> 10             raise ValueError('Marks out of range')
         11     finally:
         12         print('Bye')


        ValueError: Marks out of range
```

```
In [7]: def main():
            '''
            Objective: To illustrate the use of raise and finally clauses
            Input Parameter: None
            Return Value: None
            '''
            marks = 110
            try:
                if marks < 0 or marks > 100:
                    raise ValueError('Marks out of range')
            except:
                pass
            finally:
                print('Bye')
            print('Program continues after handling exception')

        if __name__ == '__main__':
            main()
```

```
Bye
```

```
Program continues after handling exception
```

1.10 Problem: Copying contents of a file to another file

```
In [ ]: import os

        def fileCopy(file1,file2):
            '''
            Objective: This method copies the contents in a file to another file
            Input Parameter: file1 - input file, file2 - output file
            Return Value: None
            '''
            '''
            Approach:
            Read input from file1, line by line and copy to file2 until
            null string is returned on reading
            '''
            f1 = open(file1, 'r')
            f2 = open(file2, 'w')
            line = f1.readline()
            while line != '':
                f2.write(line) #write the line from f1 with additional newline
                line = f1.readline()
            f1.close()
            f2.close()

        def main():
            '''
            Objective: To call function fileCopy to copy contents in a file to another file.
            Input Parameter: None
            Return Value: None
            '''
            fileName1=input('Enter the source file name: ')
            fileName2=input('Enter the destination file name : ')
            fileCopy(fileName1, fileName2)

        if __name__ == '__main__':
            main()
```

1.11 Computing moderated marks

- The file studentMarks contains the student data that includes roll number (rollNo), name (name), and marks (marks) for each student.
- The data about each student is stored in a separate line. Sample data in the file is shown below:

```
4001,Nitin Negi,75
```

```
4002,Kishalaya Sen,98
4003,Kunal Dua,80
4004,Prashant Sharma,60
4005,Saurav Sharma,88
```

- We define addPerCent as the percentage of maxMarks that should be added to the marks obtained to get the moderated marks, subject to the upper limit of maxMarks.
- The output file moderatedMarks containing moderated marks of the students

1. Open file studentMarks in read mode.
2. Open file moderatedMarks in write mode.
3. Read one line of input (line1) from studentMarks.
4. while (line1 != ''):

```
> Compute moderated marks and write one line of output in the file
```

moderatedMarks. > Read one line of input (line1) from studentMarks.

1.12 Problem: Compute moderated marks based on user input

```
In [10]: import sys
         def computeModeratedMarks(file1, file2, addPercent):
             '''
             Objective: To compute moderated marks of students
             Input Parameters: file1, file2: file names - string values
                               addPercent  numeric value
             Return Value: None
             Side effect: A new file  file2 of moderated marks is produced
             '''
             try:
                 fIn = open(file1, 'r')
                 fOut = open(file2,'w')
             except IOError:
                 print('Problem in opening the file'); sys.exit()
             line1 = fIn.readline()
             while(line1 != ''):
                 sList = line1.split(',')
                 try:
                     rollNo = int(sList[0])
                     name = sList[1]
                     marks = int(sList[2])
                 except IndexError:
                     print('Undefined Index'); sys.exit()
                 except (ValueError):
                     print('Unsuccessful conversion to int'); sys.exit()
                 maxMarks= 100
                 moderatedMarks = marks+((addPercent*maxMarks) /100)
                 if moderatedMarks > 100:
```

```
                moderatedMarks = 100
            fOut.write(str(rollNo) + ',' + name + ',' + str(moderatedMarks) + '\n')
            line1 = fIn.readline()
        fIn.close()
        fOut.close()
    def main():
        '''
        Objective: To compute moderated marks based on user input
        Input Parameter: None
        Return Value: None
        '''
        import sys
        #sys.path.append('F:\PythonCode\Ch09')
        # To compute moderated marks of students
        file1 = input('Enter name of file containing marks:')
        file2 = input('Enter output file for moderated marks:')
        addPercent = int(input('Enter moderation percentage:'))
        computeModeratedMarks(file1, file2, addPercent)
    if __name__=='__main__':
        main()
```

```
Enter name of file containing marks:studentMarks
Enter output file for moderated marks:moderatedMarks
Enter moderation percentage:2
```

```
In [9]: f=open('g.txt','r')

        line=f.readline()
        count=0
        while line !='':
            lineList=line.split()
            count+=len(lineList)
            line=f.readline()
        print(count)
        f.close()
```

```
4
```