Files and Exceptions

Lecture 9

Centre for Data Science, ITER
Siksha 'O' Anusandhan (Deemed to be University),
Bhubaneswar, Odisha, India.

Contents

Introduction

- Programs that we have developed so far take data from the user in an interactive manner.
- Such data remain in memory only during the lifetime of the program.
- Often we want to store data permanently, in the form of files that usually reside on disks, so that it is available as and when required.
- By default, Python provides a standard input file and a standard output file to deal with the transitory data.

Introduction (Cont.)

- The standard input file is read from the keyboard, and the standard output file is displayed on the screen.
- Apart from these standard input/output files, we can also create files on disks that store data permanently for subsequent use.
- Files provides a means to store data permanently.
- A file is a stream of bytes, comprising data of interest.
- Before performing a read or write operation in a file, we need to open the file.

File Handling

- Built-in function **open()** is used to open a file.
- This function **open()** takes the name of the file as the first argument and the mode for accessing the file as The second argument.
- A file may be opened in any of the three modes: r (read), w (write), and a (append).
- Read mode is used when an existing file is to be read.
- Write mode is used when a file is to be accessed for writing data in it.

File Handling (Cont.)

- append mode allows us to write into a file by appending contents at the end of the specified file.
- While opening a file in the read mode, if the specified file does not exist, it would lead to an error.
- while opening a file in write mode, if the specified file does not exist, Python will create a new file.
- while opening a file in write mode, if the specified file already exists, this file gets overwritten
- while opening a file in append mode, if the specified file does not exist, a new file is created.

File Handling (Cont.)

- The absence of the second argument in the open function sets it to default value 'r' (read mode).
- A file may be opened using the following syntax
  $f = open(file\_name, access\_mode)$
  >>> f = open('PYTHON', 'w')
- By default, the system creates a file PYTHON in the default working directory.
- we use write() function to write some text in the file PYTHON

  >>> f.write('failure is a part of success')

File Handling (Cont.)

- Since the file is opened in write mode, Python disallows read operation on it by flagging an error as shown below:

```
>>> f.read()

Traceback (most recent call last):
        File "<pyshell#2>", line 1, in <module>
            f.read()
io.UnsupportedOperation: not readable
```

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

```
>>> f.close()
```

File Handling (Cont.)

- The function close also saves a file, which was opened for writing.
- To read the contents of the file f, we open the file again, but this time in read mode

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

- The f.read() reads the entire contents of the file but The f.read(4) reads only 4 bytes of contents of the file.
- Consider following programs (a) and (b)

File Handling (Cont.)

(a).
```
f=open('PYTHON','w')
f.write('failure is a part of success')
f = open('PYTHON', 'r')
print(f.read(4))
f.close()
```

(b).
```
f=open('PYTHON','w')
f.write('failure is a part of success')
f = open('PYTHON', 'r')
print(f.read())
f.close()
```

- The output of (a) is **fail** and output of program (b) is **failure is a part of success**

File Handling (Cont.)

- We use the function tell() to know the current position while reading the file object f.
  >>> f.tell()
- we use function readline() to read one line at a time from the file.
  >>> f.readline()
- Note that the readline function reads a stream of bytes beginning the current position until a newline character is encountered.
- We use function seek() to reach desired position in a file
  >>> f.seek(0)
  0
- The function seek returns the new absolute position.

File Handling (Cont.)

- The function readlines() reads and returns all the remaining lines of the file in the form of a list.
  >>> f.readlines()
- Just like the readlines function discussed above, the function writelines takes a list of lines to be written in the file as an argument
  >>> f.writelines()

File Handling (Cont.)

- The output of following program is
  we either choose the pain of discipline
  or
  the pain of regret

```
f = open('PYTHON', 'w')
description =['we either choose the pain of discipline \n', 'or\n'
'the pain of regret\n']
f.writelines(description)
f.close()
f = open('PYTHON', 'r')
print(f.read())
f.close()
```

File Handling (Cont.)

- Suppose if we wish to copy the contents of a text file, say, PYTHON in another file PythonCopy. For this purpose, we open the source file PYTHON in read mode and the output file PythonCopy (yet to be created) in write mode, read the data from the file PYTHON, and write it into the file PythonCopy as follows:

```
>>> f1 = open('PYTHON', 'r')
>>> f2 = open('PythonCopy', 'w')
>>> data = f1.read()
>>> f2.write(data)
>>> f1.close()
>>> f2.close()
```

File Handling (Cont.)

- Note that if an application requires the file to be opened in both read and write mode, 'r+' mode can be used while opening it.
- If we wish to put the new content at the end of previously existing contents in a file, we need to open the file in append ('a') mode as shown below:

```
>>> f = open('PYTHON', 'a')
>>> f.write('simplicity is the ultimate sophistication')
>>> f.write('You think you can or you can't, you are right')
>>> f.close()
>>> f = open('PYTHON', 'r')
>>> f.read()
>>> f.close()
```

File Handling (Cont.)

- To ensure that the contents written to a file have been saved on the disk, it must be closed. While we are still working on a file, its contents may be saved anytime using the flush function:

```
>>> f=open('file2','w')
>>> f.write('"Believe in yourself"')
>>> f.flush()
```

Writing Structures to a File

- To write a structure such as a list or a dictionary to a file and read it subsequently, we use the Python module pickle as follows:

```
import pickle
def main():
    '''
    Objective: To write and read a list and a dictionary to
    and from a file
    Input Parameter: None
    Return Value: None
    '''
    f=open('file3','wb')
    pickle.dump(['hello','world'],f)
    pickle.dump({1:'one', 2:'two'},f)
    f.close()
```

Writing Structures to a File (cont.)

```
f=open('file3','rb')
value1=pickle.load(f)
value2=pickle.load(f)
print(value1,value2)
f.close()
if __name __=='__main __':
    main()
```

- the output of above programe is:
  ['hello', 'world'] {1: 'one', 2: 'two'}
- dump(): to convert a structure to byte stream and write it to a file
- load(): to read a byte stream from a file and convert it back to the original structure

Errors and Exceptions

- Errors occur when something goes wrong.
- The errors in Python programming may be categorized as syntax errors and exceptions.
- A syntax error occurs when a rule of Python grammar is violated.
- some Example of syntex error :
  >>> print('Hello)
  >>> for i in range(0, 10)
  indentation error
- In contrast to a syntax error, an exception occurs because of some mistake in the program that the system cannot detect before executing the code as there is nothing wrong in terms of the syntax, but leads to a situation during the program execution that the system cannot handle.

Errors and Exceptions(Cont.)

- Some common Exceptions:

1.Name error

```
>>> marks = Input('Enter your marks')
>>> print(price)
```

2.TypeError

```
>>> 'sum of 2 and 3 is ' + 5
```

3.ValueError

```
>>> int('Hello')
```

4.ZeroDivisionError

```
>>> 78/(2+3-5)
```

5.OSError

```
>>> f = open('passwordFile.txt')
```

6.IndexError

```
>>> colors = ['red', 'green', 'blue']
>>> colors[4]
>>> colors[2][4]
```

Errors and Exceptions(Cont.)

- 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.
- Remember:
  syntax error: violation of Python grammar rule
  exceptions: errors that occur at execution time

Handling Exceptions using try. . . except

- Exceptions that would result in abrupt termination of program execution are known as unhandled exceptions.
- To prevent a program from terminating abruptly when an exception is raised,We use the try. . . except clause.
- try block comprises statements that have the potential to raise an exception
- except block describes the action to be taken when an exception is raised.
- We can also specify a finally block in the try. . . except clause, which is executed irrespective of whether an exception is raised.
- 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

Errors and Exceptions(Cont.)

```
def main():
    '''
    Objective: To illustrate the use of raise and finaly 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()
```

Errors and Exceptions(Cont.)

- The OUTPUT of above Program is:
  bye
  program continues after handling exception

Errors and Exceptions(Cont.)

```
def main():
    '''
     Objective: To open a file for reading
     Input Parameter : None
     Return Value: None
    '''
     try:
          f=open('Temporary _file', 'r')
    except IOError:
          print('Problem with Input Output.....')
    print('Program continues smoothly beyond try...except block')
if __name __=='__main __':
     main()
```

Errors and Exceptions(Cont.)

- The OUTPUT of above Program is:
  Problem with Input Output.....
  Program continues smoothly beyond try...except block

Errors and Exceptions(Cont.)

```
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 inputs: ValueError')
        if weight=='': weight=None
```

Errors and Exceptions(Cont.)

```
        try:
            weight=float(weight)
        except ValueError:
            print('Invalid inputs: ValueError')
        assert price>= 0 and weight>= 0
        result=price/weight
    except TypeError:
        print('Invalid inputs: ValueError')
    except ZeroDivisionError:
        print('Invalid inputs: ZeroDivisionError')
    except:
        print(str(sys.exc _info()))
    else:
        print('Price per unit weight: ', result)
if __name __=='__main __':
    main()
```

Errors and Exceptions(Cont.)

- The OUTPUT of above Program is:
- enter price of item purchased: 20
  Enter weight of item purchased: 0
  Invalid inputs: ZeroDivisionError
- enter price of item purchased: -20
  Enter weight of item purchased: 0
  $(< class' AssertionError' >, AssertionError(), <$
  $tracebackobject at 0x000002B6FA61E400 >)$
- enter price of item purchased: -20
  Enter weight of item purchased: 10
  $(< class' AssertionError' >, AssertionError(), <$
  $tracebackobject at 0x000002B6FA60CA80 >)$
- enter price of item purchased: 20
  Enter weight of item purchased:
  Invalid inputs: ValueError

File Processing Example

- We are given a file named **studentMarks**. This file 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 and the individual pieces of information rollNo, name, and marks are separated by commas.As 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. To carry out the moderation, we prompt the user to enter the moderation percentage (addPerCent) and produce another file **moderatedMarks** containing moderated marks of the students.

File Processing Example

- We describe this task in the form of a pseudocode:
  1.Open file studentMarks in read mode while checking for any errors.
  2.Open file moderateMarks in write mode while checking for any error.
  3.Read one line of input(line1) from studentMarks.
  4.while(line !=”):
  Retrieve the values of Roll No., name, and marks from the line1 while checking for any errors.
  Compute moderated marks and write one line of output in the file moderateMarks
  Read one line of input(line1) from studentMarks.
- The complete script is given below:

File Processing Example

- The complete script is given below:
- When we execute the following program and enter 3 as value of addPercent, the system outputs the contents of the file moderatedMarks as follows:
  4001,Nitin Negi,78.0
  4002,Kishalaya Sen,100
  4003,Kunal Dua,83.0
  4004,Prashant Sharma,63.0
  4005,Saurav Sharma,91.0

File Processing Example

```
import sys
def computemoderatemarks(file1,file2,addpercent):
    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()
```

File Processing Example

```
        except (ValueError):
            print('Unsuccessful conversion to int');sys.exit()
        maxmarks=100
        moderatemarks=marks+((addpercent*maxmarks)/100)
        if moderatemarks>100:
            moderatemarks=100
        fout.write(str(rollno)+','+name+','+str(moderatemarks)+'\n')
        line1=fin.readline()
fin.close()
fout.close()
```

File Processing Example

```
def main():
    import sys
    sys.path.append('F:\pythoncode \ch9')
    file1=input('enter name of file containing marks: ')
    file2=input('enter output file for moderated marks: ')
    addpercent=int(input('enter moderation percentage: '))
    computemoderatemarks(file1,file2,addpercent)
if __name__ =='__main__':
    main()
```

File Processing Example)

In our next program, we wish to compute monthly wages to be paid to employees in an organization. The input data has been provided to us in two files named empMaster and empMonthly. The first file empMaster contains permanent data about employees (also called master data) that include employee id (empID),employee name (empName), and hourly wages (hrlyWages) as follows:
1001,Vinay Kumar,30
1002,Rohit Sen,35
1003,Vinita Sharma,28
1004,Bijoy Dutta,35

File Processing Example)

- The second file empMonthly contains monthly information (often called transaction data) about employees. It stores two pieces of information about each employee, namely, employee id (tEmpID) and the number of hours worked (hrsWorked) as follows:
  1001,245
  1002,0
  1003,0
  1004,240
- The pseudocode is given below

File Processing Example)

1.Open files **empMaster** and **empmonthly** in read mode while checking for any errors.
2.Open file **monthlyWages** in write mode while checking for any errors.
3.Read one line of input(line1) from **empMaster**
4.while (line1 !=”):
(a) Retrieve value of empID and hrlyWages from the line1 while checking for any errors.
(b) Read one line of inputs(line2) from empMonthly.
(c) Retrieve the value of tEmpID and hrsWorked from line2 while checking for any errors.
(c) check that empID in empMaster and tEmpID in the empMonthly match
(d) compute monthly wages and write one line of output in file **monthlyWages**
(d) Read one line of input(line1) from empMaster.

File Processing Example)

The complete script is given below On the execution of the following program program, the system creates an output file monthlyWages with the following content:
1001,7350
1002,0
1003,0
1004,8400

File Processing Example)

```
import sys
def generatesalary(file1,file2,file3):
    try:
        fmaster=open(file1,'r')
        ftrans=open(file2,'r')
        fwages=open(file3,'w')
    except IOError:
        print('problem with opening the file');sys.exit()
    line1=fmaster.readline()
    while((line1)!=''):
        slist1=line1.split(',')
```

File Processing Example)

```
try:
    empID=int(slist[0])
    hrlywages=int(slist1[2])
except IndexError:
    print('undefined index');sys.exis()
except (ValueError, TypeError):
    print('unsuccesssful convertion to int'); sys.exis()
line2=fTrans.readlines()
sList2=line2.split(',')
```

File Processing Example)

```
        try:
            tEmpid=int(slist2[0])
            hrsworked=int(slist2[1])
        except IndexError:
           print('undefined index');sys.exis()
        except (ValueError,TypeError):
            print('unsuccessful conversion to int');sys.exit()
        if empId==tEmpId:
            fwages.write(str(empId)+','+
str(hrlywages*hrsworked)+'\n')
        line1=fmaster.readline()
    fmaster.close()
    ftrans.close()
    fwages.close()
```

File Processing Example)

```
def main():
    import sys
    sys.path.append('F:09')
    file1=input('enter name of file containing hourly rate: ')
    file2=input('enter name of file containing hours worked: ')
    file3=input('enter output file for salary generation : ')
    generatesalary(file1,file2,file3)
if __name__=='__main__':
    main()
```

Conclusion

- A file is a stream of bytes, comprising data of interest. Built-in function open() is used for opening a file. It returns a file object. This function takes the name of the file as the first argument. The second argument indicates mode for accessing the file.
- A file may be opened in any of three modes: r (read), w (write), and a (append). Read mode is used when an existing file is to be read. Write mode is used when a file is to be accessed for writing data in it. Append mode allows one to write into a file by appending contents at the end. If the specified file does not exist, a file is created.
- The absence of the second argument in open function sets it to default value 'r' (read mode).

Conclusion

- When an existing file is opened in write mode, previous contents of the file get erased.
- The functions read and write are used for reading from and writing into the file, respectively. To use read/write function, we use the following notation: name of the file object, followed by the dot operator (.), and followed by the name of the function.
- Function close is used for closing the file. 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.
- Function **tell** yields current position in the file.

Conclusion

- The readline function reads a stream of bytes beginning the current position until a newline character is encountered. Read operation on a file whose all the contents have been read will return a null string.
- Function seek()is used for accessing the file from a particular position.
- Function readlines() returns all the remaining lines of the file in the form of a list of strings. Each string terminates with a newline character.
- Function writelines takes a list of lines to be written to the file as an argument.

Conclusion

- Pickling refers to the process of converting a structure to a byte stream before writing to the file. The reverse process of converting a stream of bytes into a structure is known as unpickling.
- Python module pickle is used for writing an object such as list or dictionary to a file and reading it subsequently. Function dump of the pickle module performs pickling. Function load of the pickle module performs unpickling.
- A syntax error occurs when a rule of Python grammar is violated.

Conclusion

- The exception occurs because of some mistake in the program that the system cannot detect before executing the code as there is nothing wrong in terms of the syntax, but leads to a situation during the program execution that the system cannot handle.
- These errors disrupt the flow of the program at a run-time by terminating the execution at the point of occurrence of the error.
- NameError exception occurs whenever a name specified in the statement is not found globally.
- TypeError exception occurs when an operation in an expression is incompatible with the type of operands.
- ValueError exception occurs whenever an invalid argument is used in a function call.
- ZeroDivisionError exception occurs when we try to perform numeric division in which denominator happens to be zero.
- IOError exception occurs whenever there is any error related to input or output.

Conclusion

- IndexError exception occurs whenever we try to access an index out of the valid range.
- try. . . except clause is used for handling the exception. Whereas a try block comprises statements, which have the potential to raise an exception, except block describes the action to be taken when exception is raised. In the except clause, we may specify a list of exceptions and the action to be taken on occurrence of each exception.
- finally block is associated with try. . . except clause and is executed irrespective of whether an exception is raised.
- Details of the exception raised by Python can be accessed from the object: sys.exc _info().

References

[1] Python Programming: A modular approach by Taneja Sheetal, and Kumar Naveen, *Pearson Education India, Inc.*, 2017.

Any question ?

Any question ?