UNIT V FILES, MODULES, PACKAGES

Files and exception: text files, reading and writing files, format operator, command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file

5.1 FILES

- File is a named location on disk to store related information.
- There are two types of files:
  - Text File
  - Binary File
- Text File are sequence of lines,where each line includes a sequence a sequence of characters.Each line is terminated with a special character,called EOL or End Of Line character.
- Binary files is any type of file other than a text file.

5.1.1 FILE OPERATIONS (4)

1. open()
2. read()
3. write()
4. close()

OPENING A FILE

The *open* Function

This function creates a **file** object, which would be utilized to call other support methods associated with it. **Syntax** file object = open("file_name" ,[mode])

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- Here is a list of the different modes of opening a file −

Modes	Description
R	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
Rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at

	the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Example.py

```
fn=open('D:/ex.txt','r')
```

READING A FILE

- This method helps to just view the content of given input file.
- Syntax: file variable=open(“filename”,’r’)

Methods used in reading a file

1. Size of data()
2. tell() and seek()
3. readline()
4. readlines()
5. File object attributes
6. Using loops
7. Handle

Size of Data: (using read())

- This read() specifies the size of data to be read from input file.
- If size not mentioned,it reads the entire file & cursor waits in last position of file.

Tell() & seek()

- Tell() method displays the current position of cursor from the input file.

- Seek() takes an argument,and moves the cursor to the specified position which is mentioned as argument.
- Syntax: print(f.tell())
  f.seek(5)

readline()

- This method is used to read a single line from the input file.
- It doesn't takes any argument.
- Syntax: f.readline()

readlines()

- This method is used to read and display all the lines from the input file.
- Syntax: f.readlines()

File Object Attributes

- Once a file is opened and you have one *file* object, you can get various information related to that file.
- Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example.py

```
file=open('D:/ex.txt','wb')
print(file.name) print(file.closed)
print(file.mode)
```

Output:

```
D:/ex.txt
False wb
```

Handle

- This is used to manipulate the file.
- If file open is successful, then OS returns a file handle.

Example.py

```
fn=open('D:/e.txt','w')
print(fn) Output:
<_io.TextIOWrapper name='D:/e.txt' mode='w' encoding='cp1252'>
```

Example2.py (Counting lines in a file)

```
fn=open('D:/e.txt') count=0 for i in fn:
   count=count+1 print(count)
Output:
2
```

Example.py (If statement in for loop)

```
fn=open('D:/e.txt') for i in fn:     if
i.startswith('This'):
      print(i)
```

Output:
This is a magic.

Example.py (prompt)

```
fn=input("enter
the filename:")
a=open(fn) count=0 for i
in a:     if
i.startswith('This'):
count=count+1
print(count) Output: enter
the filename:D:/e.txt
1
```

WRITING A FILE

Without using methods:

- To write a file, we have to open it with mode 'w' as a second parameter:
- Syntax: f.open('filename','mode')

```
>>> f = open('output.txt', 'w')
>>> print(f)
<open file 'output.txt', mode 'w' at 0xb7eb2410>
```

- If the file already exists, opening it in write mode clears out the old data and starts fresh. If the file doesn't exist, a new one is created.

Methods of writing a file:

1. write() – writes a single line into the specified file. 2. writelines() – writes multiple lines into the specified file.

write()

- The write method puts data into the file.
- The *write()* method writes any string to an open file. The write() method does not add a newline character ('\n') to the end of the string −
- **Syntax :** filevariable.write(string)
- Here, passed parameter is the content to be written into the opened file.

```
>>> line1 = "This here's the wattle,\n"
>>> f.write(line1)
```

Again, the file object keeps track of where it is, so if we call write again, it adds the new data to the end.

```
>>> line2 = "the emblem of our land.\n"
>>> f.write(line2)
```

writelines()

- The writelines method puts multiple data into the file.
- The *writelines()* method writes any string to an open file.
- **Syntax :** filevariable.writelines(string)

Example.py

```
f=open('D:/ex.txt','w') f1=[‘This is my
book \n’,’I found it here’] f.writelines(f1)
f.close()
```

Output:

```
This is my book
I found it here
```

CLOSING A FILE

close()

- The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.
- **Syntax :**filevariable.close()

Example.py

```
# Open a file f =
open("input.txt", "wb")
print("Name of the file: ", f.name)

# Close opend file
f.close() Output:
Name of the file:  input.txt
```

Python program to implement all file read operations

Example.py

```
#Opens the file newly
fn=open('D:/ex.txt','r')

#reads the entire data in file print(fn.read())

#Moves the cursor to the specified position fn.seek(4)

#Reads the data from the current cursor position after seek is done
print(fn.read())

#Moves cursor to the first position fn.seek(0)
```

```
#Reads the data in mentioned size
print(fn.read(4))

#Displays the current cursor position
print(fn.tell()) fn.seek(0)

#Reads all lines from the file
print(fn.readline()) fn.seek(0)
print(fn.readlines())
```

ex.txt

```
Welcome to the world of Robotics
```

Output:

```
Welcome to the world of Robotics ome
to the world of Robotics
Welc
4
Welcome to the world of Robotics
['Welcome to the world of Robotics']
```

Other file operations

example.py (renaming a file)

```
import
os os.rename('d:/e.txt','d:/e1.txt')
```

Output:
File is renamed.

Example.py(removing a file)

```
import os
os.remove('d:/e1.txt')
```

Output:
File will deleted from the specified path.

Example.py (append mode operation)

```
#contents of file before append mode
f=open("D:/input.txt","r")
print(f.read()) f.close()

#Open in append mode s=open("D:/input.txt","a")
s.writelines("\nThis is the end")
s.close()
```

```
#Read file after writing.
f=open("D:/input.txt","r")
print(f.read()) f.close()
```

input.txt

```
Welcome to world of robotics
This will be intresting
```

Output:

```
Welcome to world of robotics
This will be intresting
Welcome to world of robotics
This will be intresting
This is the end
```

5.1.2 FILE METHODS

No	Methods with Description
1	file.close() Close the file. A closed file cannot be read or written any more.
2	file.flush() Flush the internal buffer, like stdio's fflush. This may be a no-op on some filelike objects.
3	file.fileno() Returns the integer file descriptor that is used by the underlying implementation to request I/O operations from the operating system.
4	file.isatty() Returns True if the file is connected to a tty(-like) device, else False.
5	file.read([size]) Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes).
6	file.readline([size]) Reads one entire line from the file. A trailing newline character is kept in the string.
7	file.readlines([sizehint]) Reads until EOF using readline() and return a list containing the lines. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.
8	file.seek(offset[, whence]) Sets the file's current position
9	file.tell() Returns the file's current position
10	file.truncate([size])

	Truncates the file's size. If the optional size argument is present, the file is truncated to (at most) that size.
11	file.write(str) Writes a string to the file. There is no return value.
12	file.writelines(sequence) Writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.
13	file.readable() Returns true if file stream can be read from
14	File.seekable() Returns true if file stream supports random access.
15	File.writable() Returns true if file stream can be written to

5.1.3 COMMAND-LINE ARGUMENTS

- It means an argument sent to a program during the execution itself.
- Depending upon the program,arguments may change.
- Each argument passed is stored in sys.argv which is a list. ➢ Each argument is separated using spaces.

Example.py import
sys s=len(sys.argv)
d=str(sys.argv)
print("Total number of argv passed to script is %d"%s)
print("Arguments List %s"%d) Output:
Total number of argv passed to script is 1
Arguments List ['C:/Users/Administrator/Desktop/f3.py']

5.2 EXCEPTION

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.

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

5.2.1 Errors

1. Syntax error
2. Indentation error
3. Index error
4. Name error
5. Logical error

Example.py (Syntax error)
if x<10 print(X)
Output:
Invalid Syntax
Example.py (Indentation error)
if x<10: print(X) Output:
Expected indent block (line 2)

Example.py (Index error)
a=[1,2,3,4,5]
print(a[10]) Output:
print(a[10])
IndexError: list index out of range
Example.py (Name error) x=10
print(X) Output: print(X)
NameError: name 'X' is not defined
Example.py (Logical error) i=0
while i<5: print(i) i=i+1 Output:
0
0
0 (infinite loop,since i=i+1 is placed outside while loop)

5.2.2 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 which handles the problem as elegantly as possible.
Syntax try:
You do your operations here;
..... except
ExceptionI:
If there is ExceptionI, then execute this block.
except *ExceptionII*:
If there is ExceptionII, then execute this block.
..... else:
If there is no exception then execute this block.
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.

1.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;
   ..... 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

2.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.

Syntax :

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

- ***We cannot use else clause as well along with a finally clause.***

Example.py(divide by 0)

```
try:
   x=10/0     print("Never
executed") except:     print("this is
an error message") Output: this is
an error message
```

Example.py (except statements)

```
try:
   a=int(input("Enter a :"))
b=int(input("Enter b:"))
print(a+b)     print(a/b)
except ArithmeticError:
print("Divide by 0")
```

Output:

```
Enter a :10
Enter b:0
10
Divide by 0
```

Example.py (finally)

```
try:
   f=open("D:/input.txt","r")
```

```
   x=f.read() except
IOError:
   print("Cannot find the file") except:
   print("Some other error") else:
   print("Contents of file",x) finally:
   f.close()
   print("File closed") Output:1
   Contents of file
Welcome to world of robotics
This will be intresting
This is the end
File closed
Output:2
Cannot find the file
```

User-Defined Exceptions

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

5.3 MODULES

- A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.
- Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Types of modules:

1. Pre-defined modules
2. User-defined modules

Pre-defined modules

- These modules are already defined in Python package itself.
- Example: math,random,calendar,os,sys,numpy,string etc.,

Example.py import math print(math.sqrt(16)) print(math.pi) Output:

```
4
3.1467
```

User-defined modules

- These modules are created by the user.
- User has to write function definitions in a file and give a name with an extension .py
  - Then is name used in python program along with import keyword.
- But first this module file has to be executed before the execution of python program.

Example (calc.py)

```
def add(a,b)
        c=a+b
        return c
def sub(a,b)
```

```
        c=a-b
        return c
def mul(a,b)
        c=a*b
        return c

def div(a,b)
        c=a/b
        return c

def mod(a,b)
        c=a%b
        return c
```

main.py import
calc

```
print(calc.add(4,2))
print(calc.sub(4,2))
print(calc.mul(4,2))
print(calc.div(4,2))
print(calc.mod(4,2))
```

Output:

```
6
2
8
2
0
```

1. The *import* Statement

- You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax:

  Syntax: import module1, module2, module n
  Example: import calendar

- When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

2. The *from...import* Statement

- Python's *from* statement lets you import specific attributes from a module into the current namespace.i.e only when some methods are needed from a module.

  Syntax: from modname import name1, name2, ... name n
  Example: from calc import add

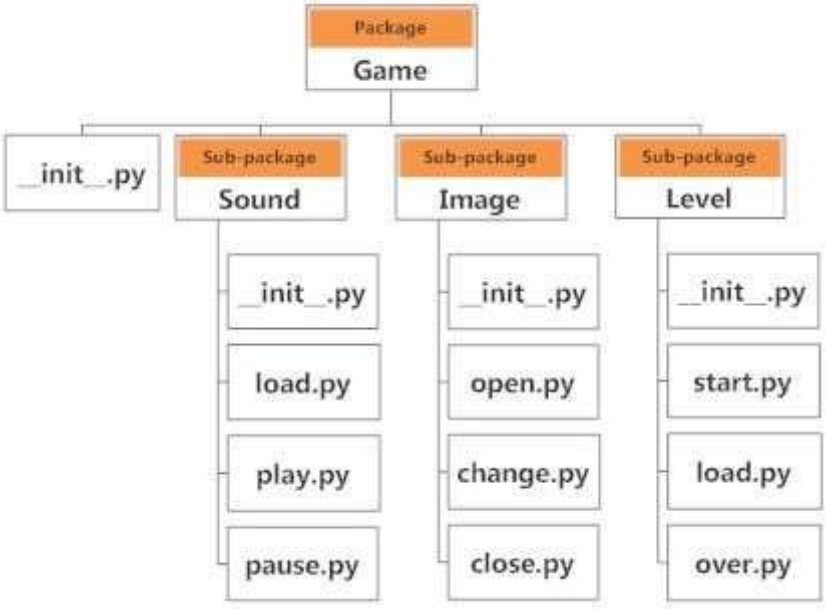
3. Aliasing Modules

➢ It is also possible to give another name to the existing modules. ➢
This is done using 'as' keyword.

Syntax: import module_name as another_name
Example: import math as m

5.4 PACKAGES

- ✓ A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and subsubpackages, and so on.
- ✓ It is a collection of many modules under a common name.
- ✓ Packages are namespaces which contain multiple packages and modules themselves.
- ✓ Each package in Python is a directory which must contain a special file called __init__.py.
- ✓ one possible organization of packages and modules.



Importing module from a package

- import modules from packages using the dot (.) operator.

EX:
import Game.Level.start #import start module
Game.Level.start.select_difficulty(2) #import the function of start module
from Game.Level import start
select_difficulty(2)

5.5 ILLUSTRATIVE PROGRAMS

5.5.1 Program to copy the contents of a file

Method 1: COPY FILE USING MODULE

```
from shutil import copyfile
source=input("enter source file name")
dest=input("enter dest file name")
copyfile("e:/source.txt","e:/dest.txt")
print("File copied successfully")
print("contents of dest file")
fn=open("e:/dest.txt","r") print(fn.read())
```

Source.Txt

```
welcome to python i
am using module
```

Output:

```
enter source file
namee:/source.txt enter dest file
namee:/dest.txt File copied
successfully contents of dest file
welcome to python
i am using module
```

Method 2: COPY FILE WITHOUT USING MODULE

```
source=input("enter source file name")
dest=input("enter dest file name")
source=open("e:/source.txt","r")
dest=open("e:/dest.txt","w") for i in
source:     dest.write(i) source.close()
dest.close() print("File copied
successfully") print("contents of dest
file") fn=open("e:/dest.txt","r")
print(fn.read()) fn.close()
```

Output:

```
enter source file nameE:/SOURCE.TXT
enter dest file namee:/dest.txt File
copied successfully contents of dest file
welcome to python i am using module
```

5.5.2 Program for counting words in a file(word count)

```
from collections import Counter fn=input("enter the file
name") c=open(fn,"r") print("no. of words in the file")
print(Counter(c.read().split())) c.close()
```

Output:

```
enter the file namee:/source.txt no.
of words in the file
Counter({'welcome': 1, 'to': 1, 'python': 1, 'i': 1, 'am': 1, 'using': 1, 'module': 1})
```