

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, Voter's age validation, Marks range validation (0-100).

FILES AND EXCEPTION

Overview

- While a program is running, its data is in memory.
- When the program ends, or the computer shuts down, data in memory disappears.

File

Explain the basic file concepts in detail. (Or) Tabulate the different modes for opening a file and explain the same. (Jan 2018) [Jan 2019] (or) Name the different access modes for opening a file and present an outline of the same. [Jan 2022]. Why does python require file handing? Explain opening files in python with all modes [May 2022]

- It is a named location on the system which stores the data for later access.
- To store data permanently, you have to put it in a file.
- Files are usually stored on permanent medium like hard drive, floppy drive or CD-ROM.
- When there are a large number of files, they are often organized into directories (also called "folders").
- Each file is identified by a unique name or a combination of a file name and a directory name.
- By reading and writing files, programs can exchange information with each other and generate printable formats like PDF.

Types of File

- Generally we divide files into two categories. They are,

a. Text File

- A text file is a simple text with sequence of characters.
- It contains readable and printable characters and whitespace, organized into lines separated by newline characters.
- It is stored on a permanent medium like a hard drive, flash memory or CD-ROM.

b. Binary File

- It is a file that contains non-readable characters in binary form.
- It is only read by the computer.

File Operations

- Python provides a built-in function for files. The four basic file related operations are,
 - Opening a File
 - Reading from a File
 - Writing to a File
 - Closing a File

Opening a File

- A file can be opened using the built-in function `open()`.
- The `open()` function takes two parameters; filename, and mode.
- To open a file, you specify its name and indicate whether you want to read or write.
- It returns a file object that contains methods and attributes about the file you opened.

Syntax

file_object = *open*("filename", "mode")

Where,

- ***file_object*** – It is the variable to add the file_object.
- ***file name*** – It defines the name of the file to be opened.
- ***mode*** – It specifies the mode. ie., Read or Write

Modes of Opening a file

The different modes of opening a file are:

- 'r' – Read mode which is used when the file is only being read.
- 'w' – Write mode which is used to edit and write new information to the file.
- 'a' – Appending mode, which is used to add new data to the end of the file.
- 'r+' – Special read and write mode, which is used to handle both actions when working with a file.
- 't' – Text Mode
- 'b' – Binary Mode

Sr.No.	Modes & Description
1.	r : Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2.	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.
3.	r+ : Opens a file for both reading and writing. The file pointer is placed at the beginning of the file.
4.	rb+ : Opens a file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.
5.	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.
6.	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.
7.	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.

8.	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.
9.	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.
10.	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.
11.	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.
12.	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

```
>>> f = open("test.txt")    # open file in current directory, Default Mode equivalent to read
>>> f = open("C:/Python27/test.txt") # specifying full path
(Or)
>>> f = open("test.dat", "w")
>>> print f
<open file 'test.dat', mode 'w' at fe820>
```

- When we print the file object, we see the name of the file, the mode and the location of the object.
- If there is no file named test.dat, it will be created.

READING AND WRITING FILES

Explain the commands used to read and write into a file with examples. [Jan 2019]

Explain about the file reading and writing operations using format operator with python code. [May 2019]

- The file object provides a set of access methods to manipulate the files. They are,
 - write()
 - read()

write()

- This method writes a string to a file.
- It does not add a new line '\n' character to the end of the string.

Syntax

file_object.write("String")

Where,

- String – It is the content to be written in opened file.

Example

```
>>> file = open("new.txt","w")
>>> file.write("Welcome to Python \n")
>>> file.write("File Concepts")
```

Output

```
Welcome to PythonFile
Concepts
```

- If the file does not exist, it creates a new file named 'new.txt'.
- If exists, it gets overwritten.

Using write along with the with() function

- can also use the write function along with the with() function:

Example:

```
with open("file.txt", "w") as f:
    f.write("Hello World!!!")
```

Output

```
Hello World!!!"
```

read()

- This method reads the contents of the file.

Syntax

file_object.read(size)

Where,

- *size* – It is the number of bytes to be read.
- If size is not specified, it reads up to the end of the file.

There are several methods available to read the contents of a file. They are,

i. read([size])

- This method reads a specified number of bytes from the open file.

Example

```
>>> file = open("new.txt","w")
>>> file.read(4)
'Welc'
```

ii. readline()

- This method reads all the characters up to and including the next newline character.

Example

```
>>> file = open("new.txt","w")
>>> print f.readline()
```

```
Welcome to Python
```

iii.readlines()

- It returns all of the remaining lines as a list of strings.

Example

```
>>> file = open("new.txt","w")
>>> print f.readlines()
['Welcome to Python', 'File Concepts']
```

Closing a File

- It is used to close a file_object using the close() method.
- Once a file_object is closed, after which no updation can be done.

Syntax

```
file_object.close()
```

Example

```
>>> file = open("new.txt", "w")
>>> print ("Name of the File:", file.name)
>>> file.close()
```

Output

Name of the File: new.txt

Python File Methods

- There are various methods available with the file object.

Method	Description
close()	Closes an opened file. It has no effect if the file is already closed.
detach()	Separates the underlying binary buffer from the TextIOBase and returns it.
fileno()	Returns an integer number (file descriptor) of the file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns True if the file stream is interactive.
read(n)	Reads at most n characters from the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Reads and returns one line from the file. Reads in at most n bytes if specified. readlines(n=-1) Reads and returns a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset, from=SEEK_SET)	Changes the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resizes the file stream to size bytes. If size is not specified, resizes to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Writes the string s to the file and returns the number of characters written.
writelines(lines)	Writes a list of lines to the file.

Example program for file concept

1. Write a python program to count number of lines, words and characters in a text file.

[Apr 2023] (8 marks)

```
file = open ("input.txt", "r")
number_of_lines = 0
number_of_words = 0
number_of_characters = 0

for line in file:
    line = line. strip("\n") #won't count \n as
    characterwords = line.split()
    number_of_lines += 1
    number_of_words += len(words)
    number_of_characters += len(line)
file. close ()
print("lines:",number_of_lines,"words:",number_of_words,"characters:",number_of_characts)
```

Output

lines:2 words:7 characters:35

Input.txt

Hello world

Come to world of python

2. Write a python function that copies a file reading and writing upto 50 characters at a time. [Apr 2023] (8 marks)

```
def copy_file_in_chunks (input_file_name, output_file_name, chunk_size=50):
    with open (input_file_name, 'r') as input_file,
    open (output_file_name, 'w') as output_file:
        while True:
            chunk = input_file. read(chunk_size)
            if not chunk:
                break
            output_file.write(chunk)

    print ('File copied successfully.')

if __name__ == '__main__':
    input_file_name = input ('Enter the name of the input file: ')
    output_file_name = input ('Enter the name of the output file: ')
    copy_file_in_chunks (input_file_name, output_file_name)
```

Output

Enter the name of the input file: input.txt

Enter the name of the output file: output.txtFile

copied successfully.

Input.txt

Hello world

Come to world of python

FORMAT OPERATOR

Write short notes on format operator. OR Discuss about the use of format operator in file processing. [Jan 2019] (Or) Explain about the file reading and writing operations using format operator with python code. [May 2019]

- The argument of write has to be a string, so if we want to put other values in a file, we have to convert them to strings.

Example

```
>>> x = 52
```

```
>>> fout.write(str(x))
```

- An alternative is to use the format operator, %.

Syntax

```
print("Format String", % (Tuple with Values))
```

- The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string.
- It contains normal text together with "argument specifiers", special symbols like "%s" and "%d".
- When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.
- The first operand is the format string, which contains one or more format sequences, which specify how the second operand is formatted. The result is a string.
- For example, it uses '%d' to format an integer, '%f' to format a floating-point number, and '%s' to format a string.

Example

```
name = "John"age = 23
```

```
print("%s is %d years old." % (name, age))
```

Output

```
"John is 23 years old."
```

- The above example uses '%s' to format a string and '%d' to format an integer.

Example

```
>>> "In %d days we made %f million %s." % (34,6.1,'dollars')In 34 days we made 6.100000 million dollars.'
```

- By default, the floating-point format prints six decimal places.
- The number of elements in the tuple has to match the number of format sequences in the string.
- Also, the types of the expressions have to match the format sequences:

Example

```
>>> "%d %d %d" % (1,2)
```

Type Error: not enough arguments for format string

```
>>> "%d" % 'dollars'
```

Type Error: illegal argument type for built-in operation

- For more control over the format of numbers, we can specify the number of digits as part of the format sequence:

```
>>> "%6d" % 62' 62'
```

```
>>> "%12f " % 6.1' 6.100000'
```

- The number after the percent sign is the minimum number of spaces the number will take up.

- If the value provided takes fewer digits, leading spaces are added.
 - If the number of spaces is negative, trailing spaces are added:
- ```
>>> "%-6d" % 62'62 '
```
- For floating-point numbers, we can also specify the number of digits after the decimal point:
- ```
>>> "%12.2f" % 6.1' 6.10'
```
- In this example, the result takes up twelve spaces and includes two digits after the decimal

COMMAND LINE ARGUMENTS

Explain about command line arguments in detail.

How to read or process command line arguments in python ? Explain with example. [Dec 2020]

- The arguments that are given after the name of the program in the command line shell of the operating system are known as Command Line Arguments.
- Python provides various ways of dealing with these types of arguments.
- The three most common are:
 - a. Using sys.argv
 - b. Using getopt module
 - c. Using argparse module

a. Using sys.argv

- ✓ The sys module provides functions and variables used to manipulate different parts of the Python runtime environment.
- ✓ This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
- ✓ One such variable is sys.argv which is a simple list structure.
- ✓ It's main purpose are:
 - **sys.argv** - is the list of command-line arguments.
 - **len(sys.argv)** - is the number of command-line arguments.
 - **sys.argv[0]** is the name of the current Python script.

Example

```
import sys
print 'Number of Arguments:', len(sys.argv)
print 'Argument List:', str(sys.argv)
```

Output

```
>>> python test.py arg1 arg2 arg3
```

Number of Arguments: 4

Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

- ✓ The arguments passed with the python script at the command line will be enclosed within sys.argv as a list.
- ✓ The command line arguments are processed as strings.

b. Using getopt module

- ✓ Python provides a get opt module that helps you parse command-line options and arguments.
- ✓ This module provides two functions and an exception to enable command line argument parsing.
- ✓ it allows both short and long options, including a value assignment.

Syntax: getopt.getopt(args, options, [long_options])

Parameters:

- ✓ **args:** List of arguments to be passed.
- ✓ **options:** String of option letters that the script want to recognize. Options that require an argument should be followed by a colon (:).
- ✓ **long_options:** List of string with the name of long options. Options that require arguments should be followed by an equal sign (=).
- ✓ **Return Type:** Returns value consisting of two elements: the first is a list of (option, value) pairs. The second is the list of program arguments left after the option list was stripped.
- ✓ The above method parses command line options and parameter list.

Example:

```
import getopt
import sys
argv = sys.argv[1:]
try:
    opts, args = getopt.getopt(argv, 'hm:d', ['help', 'my_file='])
    print(opts)
    print(args)
except getopt.GetoptError:
    print('Something went wrong!')
    sys.exit(2)
```

iii. Using arg parse module

- Using arg parse module is a better option than the above two options as it provides a lot of options such as positional arguments, default value for arguments, help message, specifying data type of argument etc.

Example:

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

ERRORS AND EXCEPTIONS**Error**

- It is a mistakes or bugs in a program done by the programmer.
- The process of finding and eliminating errors is called as debugging.
- Errors are classified into two types. They are,

Syntax Error

- Syntax errors are the most basic type of error.
- It arises when the Python parser is unable to understand a line of code.

Logical Error

- Logical errors are the most difficult to fix.
- It occurs when the program runs without crashing, but produces an incorrect result.
- The error is caused by a mistake in the program's logic.

EXCEPTION

Explain about exception handling in detail with an example. (Or) Appraise the use of try block and except block in python with syntax. (Jan 2018) (Or) Explain with an example exceptions with arguments in python. (Jan 2018) (Or) Describe how exceptions are handled in python in python with necessary examples. [Jan 2019] (Or) Explain about how exceptions are handled with examples. [May 2019] (Or) what are exceptions? Explain the method to handle them with example. [Dec 2019] (or) What is an exception? Elaborate exception handling in python. [Jan 2022]. Give a brief notes on python exception handling using try, except, raise and finally statements. [May 2022]

- An exception is an error, which occurs during the execution of a program that disrupts the normal flow of the program instructions.
- Whenever a runtime error occurs, it creates an exception.

Types of Exception

- There are two types of exceptions. They are,

Built-in Exceptions

- Illegal operations can raise exceptions when corresponding errors occur.
- Some of the built-in exceptions in Python are,
- **Zero Division Error** – It occurs when second operand of division or modulo operation is zero.

```
>>> 1/0
```

Zero Division Error: integer division or modulo by zero

- **Name Error** – It occurs when a variable is not found in local or global scope.

```
>>> 4 + spam*3
```

Name Error: name 'spam' is not defined

- **Index Error** - It is raised when index of a sequence is out of range.

```
>>> s = [1, 2, 3]
```

```
>>> s[4]
```

Index Error: list index out of range

- **Key Error** – It is raised when a key is not found in a dictionary.

```
>>> d = {1:1, 2:2}
```

```
>>> d[3]
```

Key Error: No such key in the dictionary

- **Type Error** – It is raised when a function or operation is applied to an object of incorrect type.

```
>>> '2' + 2
```

Type Error: cannot concatenate 'str' and 'int' objects

- **IO Error** – It is raised when the file is not found in the directory.

```
>>> file = open('test.txt')
```

IO Error: No such file or directory: 'test.txt'

- In each case, the error message has two parts: the type of error before the colon, and specifics about the error after the colon.

Some other exceptions are,

- **Indentation Error** - It occurs when there is incorrect indentation.
- **EOF Error** - It occurs when end of file is reached and yet operations are being performed.
- **System Error** – It occurs when the interpreter finds an internal error.
- **Import Error** – It is raised when the import statement is unable to load a module or name that cannot be found.
- **Memory Error** - It is raised when an operation runs out of memory.
- **Overflow Error** - It is raised when the result of an arithmetic operation is out of range.
- **Assertion Error** – It is raised when assert statement fails.
- **Value Error** - It is raised when a function gets argument of correct type but improper value.

User Defined Exception

- It allows the programmer to create their own exceptions by deriving classes from the standard built-in exceptions.
- In the try block, the user defined exception is raised and caught in except block.

Example

```
class Error In Code(Exception):  
    def __init__(self, data):  
        self.data = data
```

```
try:  
    raise Error In Code(2000)  
except ErrorInCode as ae:  
    print "Received error:", ae.data
```

Output

Received error: 2000

Raising an Exception

- The *raise* statement allows the programmer to force a specified exception to occur
- Raising an exception breaks current code execution and returns the exception back until it is handled.

Example

```
>>> raise Name Error('HiThere')  
  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
Name Error: Hi There
```

Example

```
try:  
    a=10  
    print a  
    raise NameError("Hello")  
except NameError as e:  
    print "An Exception Occurred"  
    print e
```

Output

10

An Exception Occurred

Hello

Handling Exceptions

- Exception handling enables you to handle errors.

a. try...except Keyword

- The *try* block is expected to produce an error and throw an exception.
- Then, the *except* block is used to catch the exception thrown by the *try* block and handle it.

Syntax

try:

Code that might throw exception

except <ExceptionType>:

Exception handler, alert the user

Example 1

```
filename = raw_input('Enter a file name: ')
```

try:

```
f = open (filename, "r")
```

except

IO Error:

```
print 'There is no file named', filename
```

Example 2

```
while True:
```

```
try: x = int(raw_input("Please Enter a Number: "))
```

```
break
```

```
except ValueError:
```

```
print "It is a Invalid Number. Try Again..."
```

The try statement works as follows.

- First, the *try block* (the statement(s) between the *try* and *except* keywords) is executed.
- If no exception occurs, the *except block* is skipped and execution of the *try* statement is finished.
- If an exception occurs during execution of the *try* block, the rest of the block is skipped.
- Then if its type matches the exception named after the *except* keyword, the *except* block is executed, and then execution continues after the *try* statement.
- If an exception occurs which does not match the exception named in the *except* block, it is passed on to outer *try* statements; if no handler is found, it is an *unhandled exception* and execution stops.

b. try...multiple except Block

- A *try* statement may have more than one *except* block, to specify handlers for different exceptions.
- At the last you can provide *else* statement. It is executed when no exception is occurred.

Write a program to handle the division by zero exception. (Nov/Dec 2022)Syntax

try:

malicious code

except Exception1:

execute code

Exception2:

execute code

....

....

except ExceptionN:

execute code

else:

try:

a=10/0
print a

except Arithmetic Error:

print "This statement is raising an exception"

else:

print "Welcome"

Output

This statement is raising an exception

c. try...finally Keyword

- If there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the *finally* block.
- *Finally* block will always be executed irrespective of the exception.

Syntax

try:

Code t That might throw exception finally:

Code which is to be executed.

Example

```
try:  
a=10/0;  
print "Exception occurred"finally:  
print "Code to be executed"
```

Output

Code to be executed

Traceback (most recent call last):

File "C:/Python27/noexception.py",
line 2, in <module>a=10/0;

Zero Division Error : integer division or modulo by zero

- The *except* and *finally* block cannot be used together with a *try* block.
- It is also not possible to use the *else* block and *finally* together with a *try* block.

Exception with an argument

- An exception can have an argument, which is a value that gives additional information about the problem.
- The contents of the argument vary from exception to exception.

You capture an exception's argument by supplying a variable in the *except* clause as follows

Example

```
try:  
b=float(56+78/0)  
except Exception, Argument:  
print 'This is the Argument\n', Argument
```

Output

This is the Argument integer division or modulo by zero

- 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.

MODULES

Explain briefly about modules with an example. (Or) What are modules in python? How will you import them? Explain concept by creating and importing a module. [Dec 2019]

- A *module* is a file that contains a collection of related functions.
- Modules in Python are simply Python files with .py extension, which contains related statements and function definitions.
- A module can define functions, classes and variables.

- A module can also include runnable code.
- To import a module, we use the import command.
- Modules are imported from other modules using the import command.

Example:

Python Module

Example

```
def add(a, b):
```

```
    """This program adds two numbers and returns the result""" result = a + b
```

```
    return result
```

The import Statement

- The import statement reads a module file and creates a module object.
- We can use any python source file as a module by executing an import statement in some other python source file.

Syntax:

```
import module1[, module2[,... moduleN]
```

- 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. The search is in this order,
- The current directory (the directory containing the input script).
- PYTHONPATH (an environment variable with a list of directory).
- The installation-dependent default directory.

Example:

```
import example
```

```
example.add(4.5,5)
```

Output:

```
9.5
```

i. The from...import Statement

- Python's from statement lets you to import specific attributes from a module into the current namespace.

Syntax:

```
from modname import name1[, name2[, ... nameN]]
```

Example:

```
from math import pi
```

```
print("The value of pi is", pi)
```

Output:

```
The value of pi is 3.141592653589793
```

ii. The from...import * Statement

- It is used to import all names (definitions) from a module into the current namespace.

Syntax:

*from modname import **

- This provides an easy way to import all the items from a module into the current namespace.

Example:

```
from math import *
```

```
print("The value of pi is", pi)
```

Syntax:

import [module] as [another_name]

Example:

```
import math as m
```

```
print(m.pi)
```

```
print(m.e)
```

Output:

```
3.14159265359
```

```
2.71828182846
```

Built-in Modules

Some of the built-in modules are,

- ***math***

It provides access to the mathematical functions defined by the C standard.

- ***random***

This module implements pseudo-random number generators for various distributions.

- ***Date time***

It supplies classes for manipulating dates and times in both simple and complex ways.

- ***sys***

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

PACKAGES

Explain python packages with an example.(or) Explain the use of packages modules in python with examples. (Nov/Dec 2022)

➤ A package is a hierarchical file directory that has collection of modules and sub-packages.

(Or)

➤ Packages are namespaces which contain multiple packages and modules themselves.

➤ It is a directory that must contain a special file called a `__init__.py` file.

➤ It can be empty file which indicates that it contains a python package.

➤ Depending on the complexity of the package, it may have more than one `__init__.py` file.

Steps to Create a Python Package

Python packages can be created as follows:

1. Create a directory and give it your package's name.
2. Insert necessary classes in it.

3. Create a `__init__.py` file in the directory.

Step 1: Create a Package Directory

➤ First we create a directory named Mobile.

Step 2: Add necessary classes

➤ Now, we create the three classes for our package.

#Sony.py

def Sony():

print ("I am a Sony Phone")

#MI.py

def MI():

print ("I am a MI Phone")

#Nokia.py

def Nokia():

print ("I am a Nokia Phone")

Step 3: Add the `__init__.py` File

➤ Finally, we create a file named `__init__.py` inside the Mobile directory and put the following code in it:

Example

from Sony import Sony

from MI import MI

from Nokia import Nokia

Importing modules from a package

➤ The modules can be imported from packages using the dot (.) operator.

➤ After adding all the classes in `__init__.py` file, we have all these classes available when we

Example

import the Mobile package.

Import Mobile Mobile.Sony()

Mobile.MI()

Output

I am a Sony Phone

I am MI Phone

PICKLING

Write short notes on pickling.

➤ Pickling is a way to convert a python object (list, dict, etc.) into a string.

➤ It is used for serializing and de-serializing a Python object structure.

Pickle Module

➤ The ***pickle*** module translates almost any type of object into a string suitable for storage in a database, and then translates strings back into objects.

➤ ***pickle.dump()*** is the method for saving the data out to the designated pickle file, usually with the .p or .pkl extension.

➤ It takes an object as a parameter and returns a string representation.

Example

```
import pickle
# import module
first grades = {'Bart':75, 'Lisa':98, 'Milhouse':80, 'Nelson':65}
f = open('gradesdict.p', 'w')      # Pickle file is newly created where foo1.py is
pickle.dump(grades, f)             # dump data to f
f.close()
```

We can unpickle the file as follows,

- First open the pickle file for reading, load the content into a new variable and close up the file.
- Loading is done through the *pickle.load()* method.

Example

```
import pickle
# import module first
f = open('gradesdict.p', 'r')      # 'r' for reading; can be omitted
mydict = pickle.load(f)            # load file content as mydictf.close()
print mydict
```

Output

```
{'Lisa': 98, 'Bart': 75, 'Milhouse': 80, 'Nelson': 65}
```

- Pickling and Unpickling can be used only if the corresponding module Pickle is imported.

Some of the common exceptions to look out for:

- **Pickle.Pickling Error:** This exception is raised when you are trying to pickle an object that doesn't support pickling.
- **Pickle.Unpickling Error:** This exception is raised when a file contains corrupted data.
- EOF Error: This exception is raised when the end of file is detected.

NAMESPACES AND SCOPING

Explain in detail about namespaces and scoping. (Apr 2024)

- In Python, a **namespace** is a fundamental concept that refers to a container where names are mapped to objects. These names are the ones we use as variable names or function names in our code.
- Each name is unique and can only be associated with one object at a time. Python manages namespaces as dictionaries, where the key is the unique name and the value is the object it refers to.

The Concept of a Namespace

A namespace ensures that all the names defined in a program are unique and can be used without any conflict. Python implements namespaces in the form of dictionaries, which makes it incredibly efficient to manage and look up these names.

Consider this simple example:

```
# Global namespace
var1 = 5

def some_func():
    # Local namespace
    var2 = 6
    def some_inner_func():
        # Nested local namespace
        var3 = 7
```

In this code snippet, `var1` exists in the global namespace, `var2` in the local namespace of `some_func`, and `var3` in the nested local namespace of `some_inner_func`. Each variable can be accessed within its scope, but not outside of it.

Types of Namespaces

Python has several types of namespaces:

- **Built-in Namespace:** This includes all the built-in object names in Python, available at any point in a program.
- **Global Namespace:** This is created when the script starts and lasts until the script is executed. It's the namespace where global variables and functions are defined.
- **Local Namespace:** This is created when a function is called and only lasts until the function returns. It includes local names inside a function.
- **Enclosing Namespace:** This exists when dealing with nested functions, where one function is defined inside another function. The enclosing namespace is the namespace of the outer function.

Scope of Objects

The **scope** of an object in Python refers to the textual region in a program where a namespace is directly accessible. For instance, in the previous example, `var2` is only accessible within `some_func`. If you try to access `var2` outside of `some_func`, you'll encounter a `NameError` because it's not in the accessible scope.

Namespace and Scope Interaction

The interaction between namespaces and scope is governed by the LEGB rule, which stands for Local, Enclosing, Global, Built-in. This rule describes the order in which Python searches for a name's namespace:

1. **Local:** The interpreter first looks for the name in the local namespace (inside the current function).
2. **Enclosing:** If not found locally, it looks in the enclosing namespace (in the current function's enclosing functions).
3. **Global:** If not found in the enclosing namespaces, it looks in the global namespace (at the top level of the executing script).
4. **Built-in:** Finally, if not found in the global namespace, it looks in the built-in namespace (Python's built-in names).

Modifying Namespaces

Python allows for the modification of namespaces using the `global` and `nonlocal` keywords.

`global`

`nonlocal`

The `global` keyword is used to declare that a variable belongs to the global namespace, while `nonlocal` is used for variables in enclosing namespaces.

For example:

```
count = 5
def some_method():
    global count
    count = count + 1
    print(count)
some_method()
```

Here, `some_method` modifies the global variable `count` because it's declared as global within the function.

ILLUSTRATIVE PROGRAMS1. DATE AND TIME

Write a python to display the current date and time. (Jan 2018)

```
import date time
now = datetime.datetime.now()
print ("Current date and time:")
print (now.strftime("%Y-%m-%d %H:%M:%S"))
```

Output

Current date and time:
2018-07-19 11:14:57

1. WORD COUNT

Design a python code to count the number of words in a python file. [May 2019]

Write a python program to count the number of words in a text file.[Dec 2019]

Write a python program to count words in a sentence using spilt () function.[May 2022]

Demo.txt

VRS Mech CSE Python VRS EEE Python Mech VRS

Program

```
file = open("demo.txt")
count = {}
for word in file.read().split():
    if word not in count:
        count[word] = 1
    else:
        print(count)file.close()
```

Output

```
count[word]+=1
{'VRS':4, 'Mech':2, 'Python':2, 'CSE':1, 'EEE':1}
```

2. COPY FILE

How to merge multiple files in to a new file using python. [Dec 2019]

```
file1 = input("Enter the source file to be copied:")
file2 = input("Enter the destination file name:")
fr = open(file1, "r")
fw = open(file2, "w") for line
```

```
in fr.readlines():
fw.write(line)
fr.close()
fw.close()
print("File Copied")
```

Output:

Enter the source file to be copied: Demo.txtEnter
the destination file name: Vrs.txt
File Copied

3. STUDENT MARK RANGE VALIDATION

```
try:
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
total=sub1+sub2+sub3+sub4+sub5
avg=total/5
if avg>=91 and avg<=100:
print("Your Grade is S")
elif avg>=81 and avg<91:
print("Your Grade is A")
elif avg>=71 and avg<81:
print("Your Grade is B")
elif avg>=61 and avg<71:
print("Your Grade is C")
elif avg>=51 and avg<61:
print("Your Grade is D")
elif avg>=40 and avg<51:
print("Your Grade is D")
elif avg>=0 and avg<40:
print("Your Grade is U")
else:
print("Invalid Input!")
except ValueError:
print("Enter valid Marks")
```

Output:

Enter marks of the first subject: 10
Enter marks of the second subject: 20
Enter marks of the third subject: 30
Enter marks of the fourth subject: 10
Enter marks of the fifth subject: 15
Your Grade is U

4. VOTER'S AGE VALIDITY

Program:

```
try:
age = int(input("Enter your age: "))
if age>=18:
```

```
print("You are Eligible for voting")
else:
print("You are not Eligible for voting")
except:
print("Age must be a valid number")
```

Output 1:

```
Enter your age: 19
You are Eligible for voting
```

Output 2:

```
Enter your age: ABC
Age must be a valid number
```

5. Print all numbers present in the text file and also print the number of blank spaces in that file.[Dec 2020]

```
fname = input("Enter file name: ")
with open(fname, 'r') as f:
for line in f:
words = line.split()
for i in words:
for letter in i:
if(letter.isdigit()):
print("Digits are",letter)
file = open("Sample.txt", "r")
count = 0
while True:
char = file.read(1)
if char.isspace():
count += 1
if not char:
break
print("Blank Space",count)
Sample.txt
hello world
hello5
hello6
CSE
```

Output:

```
Enter file name: Sample.txt
Digits are 5

Digits are 6
Blank Space 4
```

6. Write a function called sed that takes as arguments a pattern string, a replacement string,

and two filenames; it should read the first file and write the contents into the second file (creating it if necessary). If the pattern string appears anywhere in the file, it should be replaced with the replacement string. If an error occurs while opening, reading, writing or closing files, your program should catch the exception, print an error message, and exit. [Dec 2020] (or) Explain opening and closing of files in python using examples. (Nov/Dec 2022)

```
from future_____
import print_function,
division
def sed(pattern, replace, source, dest):
    fin = open("Sample.txt", 'r')
    fout = open("Test.txt", 'w')
    for line
    in fin:
        line = line.replace(pattern, replace)
        fout.write(line)
    fin.close()
    fout.close()
    def main():
        pattern = 'pattern'
        replace = 'replace'
        source = 'sed_tester.txt'
        dest = source + '.replaced'
        sed(pattern, replace, source, dest)
        print("File Copied Successfully")
    if _____name_____ == ' main ':
        main()
```

Sample.txt

hello world hello5hello6

CSE

Test.txt

hello world hello5hello6

CSE

Output:

File Copied Successfully

7. Write a program that reads a file and builds a histogram of the words in the file. [Dec 2020]

import

collections

import re

with open("./Test.txt", 'r') as f:

txt = f.read()

letters = re.findall("[a-zA-Z]", txt)

counts = collections.Counter(letters)

for i in 'abcdefghijklmnopqrstuvwxyz':

if i in counts:

print(f'{i} | {'*' * counts[i]}')

else: print(f'{i} | ")

Test.txt

Write a program that reads a file and builds a histogram of the words in the file.

Output:

```
a | *****b | *
c |
d | *****
e | *****f | ***
g | **
h | *****
i | *****j |
k |
l | ***m | **
n | **
o | *****p | *
q |
r | *****s |
****
t | *****u | *
v |
w | *x |
y |
z |
```

8. Write a program to concentrate the contents of two files into a single file. Get the input for

Two files from the user and concentrate it. [Jan 2022]

```
import shutil;
print("Enter 'x' for exit.");
filename1 = input("Enter first file name to merge: ");
if filename1 == 'x':
    exit();
else:
    filename2 = input("Enter second file name to merge: ");
    filename3 = input("Create a new file to merge content of two file inside this file: ");
    print();
    print("Merging the content of two file in",filename3);
    with open(filename3, "wb") as wfd:
        for f in [filename1, filename2]:
            with open(f, "rb") as fd:
                shutil.copyfileobj(fd, wfd, 1024*1024*10);
    print("\n Content merged successfully.!");
    print("Want to see ? (y/n): ");
    check = input();
    if check == 'n':
        exit();
    else:
        print();
        c = open(filename3, "r");
        print(c.read());
        c.close();
```

```
codes.txt
Hai Welcome
Hello world
cracker.txt
Good Morning
How are you
```

Output:

```
Enter 'x' for exit.
Enter first file name to merge: codes.txt
Enter second file name to merge: cracker.txt
Create a new file to merge content of two file inside this file: sample.txt
Merging the content of two file in sample.txt
Content merged successfully.!Want
to see? (y/n): Y
Hai Welcome
Hello world
Good Morning
How are you
```

9. Write a program to display the contents of a file by performing split operation whenever a comma is encountered in a file. (Nov/Dec 2022)

```
test_str = 'geeksforgeeks, is, best, for, geeks'
print("The original string is : " + str(test_str))
res = test_str.replace(", ", ", ").split()
print("The strings after performing splits : " + str(res))
```

Output

```
The original string is : geeksforgeeks, is, best, for, geeks
The strings after performing splits : ['geeksforgeeks', ',', 'is', ',', 'best', ',', 'for', ',', 'geeks']
```

10. Write a Python program to implement stack operations using modules. [APR 2023]

class Node:

```
def __init__(self, value):
    self.value = value
    self.next = None
class Stack:
    def __init__(self):
        self.head = Node("head")
        self.size = 0
    def __str__(self):
        cur = self.head.next
        out = ""
        while cur:
            out += str(cur.value) + "->"
            cur = cur.next
        return out
```

```

out[:2] def getSize(self):
return self.sizedef isEmpty(self):
return self.size == 0def peek(self):
if self.isEmpty():
raise Exception("Peeking from an empty stack")return
self.head.next.value
def push(self, value):
node = Node(value) node.next = self.head.next
self.head.next = node self.size += 1
def pop(self):
if self.isEmpty():
raise Exception("Popping from an empty stack")
remove = self.head.next
self.head.next = self.head.next.nextself.size -= 1
return remove.value
if __name__ == "__main__":stack = Stack()
for i in range(1, 11):
stack.push(i)
print(f"Stack: {stack}")
for _ in range(1, 6):
remove = stack.pop()
    print(f"Pop: {remove}")
print(f"Stack: {stack}")

```

OUTPUT:

```

Stack: 10->9->8->7->6->5->4->3->2->1
Pop: 10
Pop: 9
Pop: 8
Pop: 7
Pop: 6
Stack: 5->4->3->2->1

```

11. Write a program to enter a number in python and print its octal and hexadecimal equivalent (Apr2024)

- This program takes an integer input, converts it to both octal and hexadecimal formats using the oct() and hex() functions, and prints the results

```

Enter a number
num = int(input("Enter a number: "))
# Convert to octal and hexadecimal
octal = oct(num)

```

```
hexadecimal = hex(num)
# Print the results
print(f"Octal equivalent: {octal}")
print(f"Hexadecimal equivalent: {hexadecimal}")
```

OUTPUT:

Enter a number: 50
Octal equivalent: 0o62
Hexadecimal equivalent: 0x32

Explanation:

The octal equivalent of 50 is 0o62 (0o indicates octal).

The hexadecimal equivalent of 50 is 0x32 (0x indicates hexadecimal).

2 Marks

1. What is a File? (Or) Define File.

- It is a named location on the system which stores the data for later access.
- To store data permanently, you have to put it in a **file**.
- Files are usually stored on permanent medium like hard drive, floppy drive or CD-ROM.
- When there are a large number of files, they are often organized into **directories** (also called "folders").

2. What are the basic file operations available in python?

- Opening a File
- Reading from a File
- Writing to a File
- Closing a File

3. How the files are classified?

- Generally we divide files into two categories. They are,

Text File

- A text file is a simple text with sequence of characters.
- It contains readable and printable characters and whitespace

Binary File

- It is a file that contains non-readable characters in binary form.
- It is only read by the computer.

4. Write the syntax for opening a file.

```
file_object = open("filename", "mode")
```

Where,

- ***file_object*** – It is the variable to add the *file_object*.
- ***file name*** – It defines the name of the file to be opened.
- ***mode*** – It specifies the mode. ie., Read or Write

5. Write the syntax for opening a file to write in binary mode. [Jan 2022]

Syntax for opening a file

```
file_object = open("filename", "mode")
```

Syntax for writing a file

```
file_object.write("String")
```

Example:

```
f=open("sample.txt",wb)
f.write("123")f.close()
```

6. Explain the methods used to read a file.

read([size])

- This method reads a specified number of characters from the opened file.

readline()

- This method reads all the characters up to and including the next newline character.

readlines()

- It returns all the remaining lines as a list of strings in a file.

7. What is Format Operator?

- % operator is called as format operator.
- The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string.
- It contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

8. Define command line arguments. (Or) What is command line argument? [May 2019] Or How to use command line arguments in python? [Dec 2019] Or

How do you use the command line arguments to give input to the program? [May 2023]

- Python provides a getopt module that helps you parse command-line options and arguments.
- The Python sys (system) module provides access to any command-line arguments via the sys.argv.

They are accessed by the module name followed by the period operator and the functionality name.

Example

```
import sys
print 'Number of Arguments:', len(sys.argv)
print 'Argument List:', str(sys.argv)
```

Output

```
>>> python test.py arg1 arg2 arg3
Number of Arguments: 4 Argument
List: ['test.py', 'arg1', 'arg2', 'arg3']
```

9. What is sys.argv?

- sys.argv is the list of command line arguments passed to the Python program.
- argv represents all the items that come along via the command line input.
- It's basically an array holding the command line arguments of our program.

10. Define Error

- It is a mistakes or bugs in a program done by the programmer.
- The process of finding and eliminating errors is called as debugging.

11. What are the types of error?

- Errors are classified into two types. They are,
- Syntax Error – It arises when the Python parser is unable to understand a line of code.
- Logical Error - It is caused by a mistake in the program's logic.

12. What are Exception? (Nov/Dec 2022)

- An exception is an error, which occurs during the execution of a program that disrupts the normal flow of the program instructions.
- Whenever a runtime error occurs, it creates an exception.

13. List the types of exception.

There are two types of exceptions. They are,

- Built-in Exceptions - Illegal operations can raise exceptions when corresponding errors occur.
- User Defined Exception - It allows the programmer to create their own exceptions by deriving classes from the standard built-in exceptions.

14. What is raising an exception?

- The *raise* statement allows the programmer to define a new exception to occur.
- An exception can be a string, a class or an object.
- Raising an exception breaks current code execution and returns the exception back until it is handled.

15. What is Exception Handling? (Or) Define Exception Handling.

- Exception handling enables you to handle errors.
- In python, *try* block organizes a block of code that is expected to produce an error and throw an exception.
- Then, the *except* block is used to catch the exception thrown by the *try* block and handle it.

16. List some built-in exceptions in python.

- *Zero Division Error* – It occurs when second operand of division or modulo operation is zero.
- *Name Error* – It occurs when a variable is not found in local or global scope.
- *Index Error* - It is raised when index of a sequence is out of range.
- *Key Error* – It is raised when a key is not found in a dictionary.
- *IO Error* – It is raised when the file is not found in the directory

17. What is closing a file?

- It is used to close a file object using the `close()` method.
- Once a file object is closed, after which no updation can be done.

Syntax: `file_object.close()`

18. Define Package. (Or) What is Python Package? What are packages? How to import modules from a package. [Dec 2020]

- A package is a hierarchical file directory that has collection of modules and sub-packages.
- It is a directory that must contain a special file called a `__init__.py` file.
- It can be empty file which indicates that it contains a python package.
- Depending on the complexity of the package, it may have more than one `__init__.py` file.

Example

```
from Sony import Sony
from MI import MI
from Nokia import Nokia
```

Importing modules from a package

- The modules can be imported from packages using the dot (.) operator.
- After adding all the classes in `__init__.py` file, we have all these classes available when we

Example

```
import the Mobile package.  
Import Mobile  
Mobile.Sony()  
Mobile.MI()
```

Output

```
I am a Sony Phone  
I am a MI Phone
```

9. Write the steps to create a python package.

Python packages can be created as follows:

- Create a directory and give it your package's name.
- Insert necessary classes in it.
- Create a `__init__.py` file in the directory.

20. Define Pickling.

- **Pickling** is a way to convert a python object (list, dict, etc.) into a string.
- It is used for serializing and de-serializing a Python object structure.

21. What is Pickle Module?

- The **pickle** module translates almost any type of object into a string suitable for storage in a database, and then translates strings back into objects.

22. Find the syntax error in the code given: [Jan 2019]

```
While true print ('Hello World')  
While true:  
Print ('Hello World')
```

23. Categorise the different types of errors arises during programming interpret the followingpython code.[May 2019]

```
>>> import os  
>>> cwd = os.getcwd()  
>>> print cwd  
  
/home/dinsdale
```

File not found Error.No such file directory was found.

The program will produce the current working directory of
python.Ex: C:/python 27

24. What is Module? (Or) Define Module. (Or) Write short notes on Modular Design. (Jan 2018) [Jan 2019]

- A **module** is a file that contains a collection of related functions.
- Modules in Python are simply Python files with `.py` extension, which contains related statementsand function definitions.
- Modules are imported from other modules using the **import** command.

25. What is an import Statement?

- The **import** statement reads a module file and creates a module object.
- We can use any python source file as a module by executing an **import** statement in some other python source file.
- **Syntax:** `import module1[, module2[,... moduleN]`

26. What is *from...import* Statement?

- Python's ***from*** statement lets you to import specific attributes from a module into the current namespace.
- **Syntax:** *from module name import name1[, name2[, ... nameN]]*

27. What is *from...import ** Statement?

- It is used to import all names (definitions) from a module into the current namespace.
- **Syntax:** *from module name import **

Unit-5 Questions

1. Why does python require file handing? Explain opening files in python with all modes.
2. How do you use the command line arguments to give input to the program?
3. Explain the use of packages modules in python with examples.
4. What is an exception? Elaborate exception handling in python.

Program

5. Write a python program to count number of lines, words and characters in a text file.
6. Write a python function that copies a file reading and writing upto 50 characters at a time.
7. Write a program to handle the division by zero exception.
8. Write a program to display the contents of a file by performing split operation whenever a comma is encountered in a file.
9. Write a Python program to implement stack operations using modules.