SHRI G.P.M. DEGREE COLLEGE OF SCIENCE & COMMERCE

SHRI G.P.M. DEGREE COLLEGE OF SCIENCE & COMMERCE.

(COMMITED TO EXCELLENCE IN EDUCATION)

CERTIFICATE

This is to certify that Mr/Ms.____

a student of FY-BSC-IT Roll no:__________ has completed the required number of practicals in the subject of ____

as prescribed by the UNIVERSITY OF MUMBAI undermy supervision during the academic year 2023-2024.

……………….. ……………………

Prof. Incharge Principal

…………………… …..…..…………

External Examiner Course Co-ordinator

…………………….. ……………………….

Date College Stamp

Prof. Name: Abdul Ahad Naeemi	**Class/ Sem: F.Y. B.Sc. - CS / Sem – II (2023-2024)**
Course Code: USCSP202	**Subject Name: Advanced Python Programming**

Sr. No.	Date	INDEX	Pg. No.	Sign.
		Theory 1: Working with files		
1		**Practical-1:** Write a program to Python program to implement various file operations. **(IT Lab)**		
2		**Example-1:** Write a python program to read the file ‘new.txt’. **(IT Lab)**		
3		**Example 2:** Write a python program to write multiple lines into the file. **(Homework)**		
		Theory 2: Regular Expression		
4		**Practical 2:** Write a program to Python program to demonstrate use of regular expression for suitable application.		
5		**Example 1**: Write python program to print ‘my first RE program ‘and display inverted commas for each word. **(IT Lab)**		
6		**Example 2:** Write a python program to print ‘my beautiful daughter without displaying [a-e] letters in it. **(Homework)**		
		Theory 3: Threads in Python		
7		**Practical 3:** Write a Program to demonstrate concept of threading and multitasking in Python.		
8		**Example 1:** Write a Program to demonstrate concept of Single thread		
9		**Example 2:** Write a Program to demonstrate concept of multithreading.		
		Theory 4: Database in Python		
10		**Practical 4:** Write a Python Program to work with databases in Python to perform operations such as a. Connecting to database b. Creating and dropping tables c. Inserting and updating into tables.		
11		**Example 1:** Write a Python program to insert and delete a row in table.		
13		**Example 2:** Write a Python program for exception handling in database		
		Theory 5: Exception Handling		
14		**Practical 5:** Write a Python Program to demonstrate different types of exception handing**(IT Lab)**		
15		**Example 1**: Write a python program to demonstrate exception handling from ‘myfile.txt’. **(IT Lab)**		
16		**Example 2:** Write a python program to demonstrate exception handling from ‘sample.txt’. **(Homework)**		
		Theory 6: Graphical User Interface		
17		**Practical 6 :** Write a GUI Program in Python to design application that demonstrates a. Different fonts and colors, b. Different Layout Managers b. Event Handling **(IT Lab)**		
18		**Example 1:** Write a python program to create line with rectangle having canvas width =200 and height =100. **(IT Lab)**		
19		**Example 2:** write a python program for creating two lines with canvas width 60 and height 80. **(Homework)**		

20 21 22		**Theory 7: Date and Time in Python** **Practical 7:** Write Python Program to create application, which uses date and time in Python. **Example 1:** Write a Python Program for combining date and time. **Example 3:** Write a Python Program to compare two dates		
23 24 25		**Theory 8: Client Server information** **Practical 8 :** Write a Python program to create server-client and exchange basic information **Example 1:** write a python program to create a server client and exchange current-day information. **(IT Lab)** **Example 2:** write a python program to create a server-client and exchange current time information. **(Homework)**		
26 27 28		**Theory 9: Implementation of OOP concept** **Practical-9 :** Write a Python program to implement concepts of OOP such as a. Types of Methods b. Inheritance c. Polymorphism **Example 1:** Write a Python program to implement the concept of Inheritance with one base class and two child classes. **Example 2:** Write a Python program to implement polymorphism: Define methods in the child class that have the same name as the methods in the parent class.		
29 30		**Theory 10: Implementation of OOP concept** **Practical-10:** Write a program to Python program to implement concepts of OOP such as a. Abstract methods and classes b. Interfaces **Example 1:** Write a program to create a abstract class with name 'Bike' and having one method run().		

******

Theory- 1

Implementation of File Operations

Creating a File

In Python, you can create a new file using the open() function. This function takes two parameters: the file name and the mode. The mode parameter can be set to 'r' for read-only, 'w' for write-only, and 'a' for append. Depending on the mode chosen, the file will either be opened or created.

Once you have created a file, you can write to it using the write() method. This method takes a string as a parameter, which is then written to the file. Note that this method will overwrite any existing content in the file.

Opening a File

In Python, you can open an existing file using the open() function. The first parameter is the file name, and the second parameter is the mode.

The mode parameter can be set to 'r' for read-only, 'w' for write-only, and 'a' for append. If the file does not exist, the open() function will create it.

Once the file is open, you can read from it using the read() method. This method takes no parameters and returns the entire contents of the file as a string. Alternatively, you can readline() which returns one line at a time.

Modifying a File

In Python, you can modify an existing file using the write() method. This method takes a string as a parameter, which is then written to the file. Note that this method will overwrite any existing content in the file.

The append method is another way of modifying files in Python. This method adds content to the end of the file without overwriting the existing content. It can be used instead of write() if you want to keep the content that is already in the file.

The append method takes a string as a parameter, which is then added to the end of the file. Unlike the write() method, the append() method does not overwrite the existing content. Instead, it adds the string to the end of the file.

You can also use the seek() method to move the pointer to a specific location in the file. This method takes an integer as a parameter, which is the offset from the beginning of the file.

Closing a File

In Python, you can close a file using the close() method. This method takes no parameters and will close the file.

Note that once the file is closed, any changes made to the file will be lost unless the file is saved.

Practical-1: Write a Python program to implement various file operations.

Source code:

```
#Creating a file
myfile = open('myfile.txt','w')  #opens a
file in write mode

#Writing to the file
myfile.write('Hello world!\n')
myfile.write('This is my first file in
Python\n')

#Reading from the file
myfile = open('myfile.txt','r')
print(myfile.read())
print(myfile.tell())

#Appending to the file
myfile = open('myfile.txt','a+')
myfile.write('This is an appended text.\n')

myfile.seek(45,0)
print(myfile.readline())

#Closing the file
myfile.close()
```

Output:

```
Hello world!
This is my first file in Python

45
This is an appended text.
```

Example-1: Write a python program to read the file 'new.txt'.

Source code:

```
with open("new.txt","w+") as f:
    f.write("Hello World\nThis is my file")
    f.seek(0)
    print(f.read())
```

Output:

```
Hello World
This is my file
```

Example-2: : Write a python program to read the full text from file 'first.txt'.

Source code:

```
with open("file.txt","w+") as f:
    f.write("Hello World\nThis is my file")
    f.seek(0)
    print(f.read())
```

Output:

```
Hello World
This is my file
```

Example 3: Program to write the file.

Source code:

```
with open("file1.txt","w+") as f:
    f.write("Python is Awesome")
    f.seek(0)
    print(f.read())
```

Output:

```
Python is Awesome
```

Example -4: Write a python program to write into file.

Source code:

```
with open("file1.txt","w+") as f:
    f.write("File made with Python")
    f.seek(0)
    print(f.read())
```

Output:

```
File made with Python
```

COMMITTED TO EDUCATION

निर्मलस्नेह उत्तम सेवाधर्म

Example-5: Write a python program to write multiple lines into the file.

Source code:

```
with open("file.txt","w+") as f:
    f.write("""
           The ZEN of Python

    Beautiful is better than ugly.
    Explicit is better than implicit.
    Simple is better than complex.
    Complex is better than complicated.
    Flat is better than nested.
    Sparse is better than dense.""")
    f.seek(0)
    print(f.read())
```

Output:

```
       The ZEN of Python

 Beautiful is better than ugly.
 Explicit is better than implicit.
 Simple is better than complex.
 Complex is better than complicated.
 Flat is better than nested.
 Sparse is better than dense.
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. How to open a file ?
2. How to write in file ?
3. How to read the text from file ?
4. How to close a file ?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

COMMITTED TO EXCELLENCE IN EDUCATION
निर्मलस्नेह ज्ञान सेवाधर्म

Theory - 2

Regular Expressions

Regular expressions, also known as regex or regexp, are powerful tools for performing pattern matching on text. In essence, regular expressions are a tiny, highly specialized programming language embedded inside a larger language such as Python, Perl, or Java. Regular expressions are used to search, edit, and manipulate text based on patterns.

Regular expressions are commonly used for a wide variety of tasks such as validating user input, searching and replacing text, and extracting specific data from large data sets. Regular expressions are made up of a combination of literal characters and metacharacters. Metacharacters are special characters that have a specific meaning to the regex engine.

The metacharacters are:

1) The dot (.) - The dot metacharacter is used to match any character.

2) The asterisk (*) - The asterisk metacharacter is used to match zero or more occurrences of the preceding character or group.

3) The plus (+) - The plus metacharacter is used to match one or more occurrences of the preceding character or group.

4) The question mark (?) - The question mark metacharacter is used to match zero or one occurrence of the preceding character or group.

5) The pipe (|) - The pipe metacharacter is used to match either of two alternatives.

6) The backslash (\) - The backslash metacharacter is used to escape metacharacters and other special characters.

7) The brackets ([ ]) - The brackets metacharacters are used to match one of many characters.

8) The caret (^) - The caret metacharacter is used to match the beginning of a line.

9) The dollar sign ($) - The dollar sign metacharacter is used to match the end of a line.

10) The parentheses (( )) - The parentheses metacharacters are used to group characters.

11) /s: Matches any whitespace character (space, tab, newline, etc.)

12) /w: Matches any word character (a-z, A-Z, 0-9, and underscore)

13) /d: Matches any digit character (0-9)

14) /S: Matches any non-whitespace character

15) /W: Matches any non-word character

16) /D: Matches any non-digit character

17) /z: Matches the end of the string or line

There are various regex methods which are used to perform various operations on strings. The seven regex methods are:

1) findall() - The findall() method is used to find all matches of a pattern in a string.

2) search() - The search() method is used to search for a match in a string.

3) split() - The split() method is used to split a string into a list of substrings.

4) sub() - The sub() method is used to replace a pattern in a string with a specified string.

5) match() - The match() method is used to match a pattern at the beginning of a string.

6) compile() - The compile() method is used to compile a regex pattern into a regex object.

7) finditer() - The finditer() method is used to find all matches of a pattern in a string and return an iterator.

Regular expressions are a powerful tool for pattern matching and text manipulation. They are made up of literal characters and metacharacters which have special meaning to the regex engine. There are 10 metacharacters and 7 regex methods which are used to perform various operations on strings. Regular expressions can be used for a wide variety of tasks such as validating user input, searching and replacing text, and extracting specific data from large data sets.

Regular expressions are a powerful tool for pattern matching and text manipulation. They are made up of literal characters and metacharacters which have special meaning to the regex engine. There are 10 metacharacters and 7 regex methods which are used to perform various operations on strings. Regular expressions can be used for a wide variety of tasks such as validating user input, searching and replacing text, and extracting specific data from large data sets.

Regular expressions are incredibly powerful tools for searching, editing, and manipulating text. They can be used to quickly find patterns in text, to check for valid input, and to extract useful data from large datasets. They are composed of literal characters as well as special metacharacters that have a particular meaning to the regex engine.

COMMITTED TO EXCELLENCE IN EDUCATION
निर्मलस्नेह उत्तम सेवाधर्म

Practical-2: Write a program to Python program to demonstrate use of regular expression for suitable application.

Source code:

```
import re

pattern =
r"(\w{3,20}\D+\d*)@(((g|e)(mail))|yahoo)\
.com"
email = input()

if re.match(pattern, email):
    print(f"Email: {email}\n\n{'
'*((len(email)//2)-2)}|Valid|")
else:
    print(f"Email: {email}\n\n{'
'*(len(email)//2)}|Invalid|")
```

COMMITTED TO EXCELLENCE IN EDUCATION

Output:

```
Email: noteve@gmail.com

       |Valid|
```

Example-1: Write python program to print ‘my first RE program ‘and display inverted commas for each word.

Source code:

```
import re

str1 = "my first RE program"

str1 = str1.replace(" ", ",")

print(str1)
```

Output:

```
my,first,RE,program
```

Example-2: Write a python program to print 'my beautiful daughter without displaying [a-e] letters in it.

Source code:

```
import re

print(re.sub(r'[a-e]','', 'my beautiful
daughter'))
```

Output:

```
my utiful ughtr
```

निर्मलस्नेह उत्तम सेवाधर्म
COMMITMENT TO EXCELLENCE IN EDUCATION

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What is regular expression ?
2. Which module supports regular expression ?
3. What are metacharacters ?
4. What are the various regex functions ?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory - 3

Threads in Python

Threads are used in programming to allow multiple tasks to be executed in parallel. Each thread has its own memory space, so data is not shared between threads. This means that threads can be used to simplify the process of running multiple tasks at the same time, as each thread can be dedicated to a single task. Threads are also useful for improving the performance of applications, as each thread can be used to perform a different task, so the overall speed of the application is increased.

In Python, threads are implemented using the threading module. This module contains a number of functions that allow you to create, manage and control threads. The most important functions are the threading.Thread() class, which is used to create a new thread, and the threading.Lock() class, which is used to protect shared resources from being accessed by multiple threads at the same time.

To use threads in Python, you first need to create an instance of the threading.Thread() class. This class takes a function as its only argument, which is the code that will be executed by the thread. Once the thread has been created, it needs to be started by calling the start() method. Once the thread has started, it will execute the code until it is finished or until it is stopped by calling the join() method.

Threads can also be used to execute code in a specific order. This is done by creating a queue of functions that will be executed in order. Each time a thread is created, it will take the next function in the queue and execute it. This allows you to ensure that certain code is executed before other code, which can be useful for tasks such as data processing.

Threads can also be used to achieve concurrency in Python. Concurrency is when two or more threads are executing concurrently, which can be useful for applications that need to respond to multiple requests at the same time. In Python, concurrency is achieved using the threading.ThreadPoolExecutor() class, which allows you to create a pool of threads that can be used to execute code concurrently.

Threads can also be used to improve the performance of Python applications. By using multiple threads, the application can be split into multiple pieces, allowing each piece to be processed in parallel. This can result in a significant performance improvement, as the application can take advantage of multiple cores, instead of just one.

In summary, threads are an important feature of Python that can be used to improve the performance and efficiency of applications. Using threads, code can be executed in parallel, code can be executed in a specific order, and concurrency can be achieved. Threads are also useful for protecting shared resources from being accessed by multiple threads at the same time. With the help of the threading module, Python developers can easily take advantage of the power of threads in their applications.

Practical-3: Write a Program to demonstrate concept of threading and multitasking in Python.

Source code:

```
import threading
import time

starting_point = time.perf_counter()

def eat_breakfast():
    time.sleep(3)
    print("You eat breakfast")

def drink_coffee():
    time.sleep(4)
    print("You drank coffee")

def study():
    time.sleep(5)
    print("You finish studying")

x= threading.Thread(target=eat_breakfast,
args=())

x.start()

y= threading.Thread(target=drink_coffee,
args=())

y.start()

z = threading.Thread(target=study, args=())
z.start()

x.join()
y.join()
z.join()

print(threading.active_count())
print(threading.enumerate())
print(time.perf_counter()-starting_point)
```

Output:

```
You eat breakfast
You drank coffee
You finish studying
1
[<_MainThread(MainThread, started
139986127722304)>]
5.002365078777075
```

Example 1: Write a Program to demonstrate concept of Single thread

Source code:

```
import threading,time

start_time = time.perf_counter()

def eat_breakfast():
    time.sleep(1)
    print("You eat breakfast")

def drink_coffee():
    time.sleep(2)
    print("You drank coffee")

def study():
    time.sleep(3)
    print("You finish studying")

eat_breakfast()
drink_coffee()
study()

print("""
No. of threads:""",threading.active_count())

print("Thread name:",threading.enumerate())
print("Time
taken:",time.perf_counter()-start_time,"secs
")
```

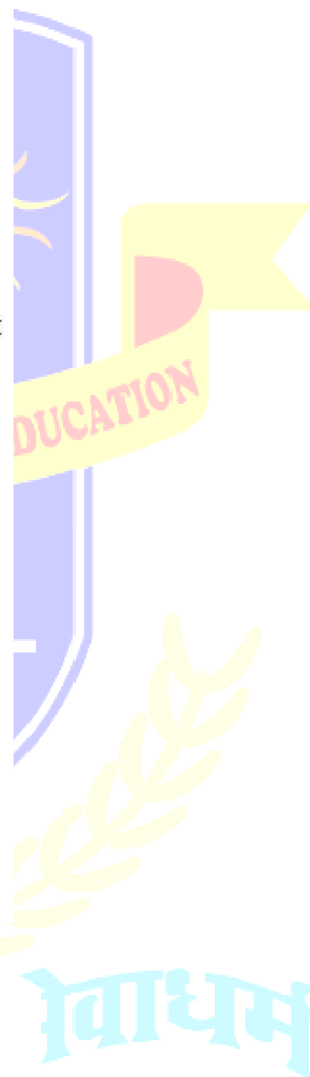
Output:

```
You eat breakfast
You drank coffee
You finish studying

No. of threads: 1
Thread name: [<_MainThread(MainThread, started 505565611256)
>]
Time taken: 12.012174461997347 secs
```

Example 2: Write a Program to demonstrate concept of multithreading.

Source code:

```
import threading
import time

starting_point = time.perf_counter()

def eat_breakfast():
    time.sleep(3)
    print("You eat breakfast")

def drink_coffee():
    time.sleep(4)
    print("You drank coffee")

def study():
    time.sleep(5)
    print("You finish studying")
    print("\nTime taken by
threads:",time.perf_counter()-starting_point
,"secs")

x= threading.Thread(target=eat_breakfast,
args=())

x.start()

y= threading.Thread(target=drink_coffee,
args=())

y.start()

z = threading.Thread(target=study, args=())
z.start()

print("No. of
threads:",threading.active_count())
```



Output:

```
No. of threads: 4
You eat breakfast
You drank coffee
You finish studying

Time taken by threads: 5.001663303002715
secs
```

Example 3: Write a Program for thread synchronization.

Source code:

```
import threading
import time

starting_point = time.perf_counter()

def eat_breakfast():
    time.sleep(3)
    print("You eat breakfast")

def drink_coffee():
    time.sleep(4)
    print("You drank coffee")

def study():
    time.sleep(5)
    print("You finish studying")

x= threading.Thread(target=eat_breakfast,
args=())

x.start()

y= threading.Thread(target=drink_coffee,
args=())

y.start()

z = threading.Thread(target=study, args=())
z.start()

x.join()
y.join()
z.join()

print("No.of
threads:",threading.active_count())
print(threading.enumerate())
print("Time
taken:",time.perf_counter()-starting_point,"
secs")
```

Output:

```
You eat breakfast
You drank coffee
You finish studying
No.of threads: 1
[<_MainThread(MainThread, started
140369210857280)>]
Time taken: 5.004070029594004 secs
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What are the threads ?
2. Which module is used to create a thread ?
3. Which method is used to count number of threads ?
4. Which method is used to find the time taken by the threads ?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 4

Database in Python

MySQL Connector/Python is a standardized database driver for Python platforms and development. This module is designed to be used with MySQL Server version 8.0 and higher and supports the Python Database API Specification v2.0. The MySQL Connector/Python is written in pure Python, and it is self-contained, meaning no additional libraries are needed for it to work.

MySQL Connector/Python provides an interface for connecting to a MySQL database by creating a connection object. The syntax for this is as follows:

connection = mysql.connector.connect(user='username', password='password', host='hostname', database='database_name')

In this statement, the user parameter is the username of the MySQL user, the password parameter is the password of the MySQL user, the host parameter is the hostname of the MySQL server, and the database parameter is the name of the database to be used. Once the connection is established, the connection object can be used to execute SQL statements against the database.

MySQL Connector/Python provides a cursor object that can be used to execute SQL statements and fetch data from the database. The syntax for creating a cursor object is as follows:

cursor = connection.cursor()

This statement creates a cursor object that can be used to execute SQL statements against the database. The cursor object provides several methods for executing SQL statements, such as the execute() method. This method can be used to execute any SQL statement, such as SELECT, INSERT, UPDATE, and DELETE.

The cursor object also provides several methods for retrieving data from the database, such as the fetchall() method. This method retrieves all the rows from the result set and returns them as a list of tuples. The fetchone() method retrieves a single row from the result set and returns it as a tuple.

Finally, the cursor object also provides a method for committing changes to the database, the commit() method. This method is used to commit any changes that have been made to the database.

Creating a Database:

MySQL Connector/Python allows users to create a database through a simple command. To create a database, the CREATE DATABASE statement is used. The syntax for this statement is as follows:

CREATE DATABASE database_name;

In this statement, the database_name parameter is the name of the database to be created.

Creating a Table:

MySQL Connector/Python provides a simple way to create a table. To create a table, the CREATE TABLE statement is used. The syntax for this statement is as follows:

CREATE TABLE table_name (column1 datatype, column2 datatype, ...);

In this statement, the table_name parameter is the name of the table to be created. The column1 and column2 parameters are the names of the columns in the table. The datatype parameters are the data types of the columns.

Insert Rows:

MySQL Connector/Python allows users to insert rows into tables. To insert rows, the INSERT INTO statement is used. The syntax for this statement is as follows:

INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

In this statement, the table_name parameter is the name of the table into which the rows will be inserted. The column1 and column2 parameters are the names of the columns into which the values will be inserted. The value1 and value2 parameters are the values to be inserted.

Update Row:

MySQL Connector/Python allows users to update rows in tables. To update a row, the UPDATE statement is used. The syntax for this statement is as follows:

UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;

In this statement, the table_name parameter is the name of the table in which the row will be updated. The column1 and column2 parameters are the names of the columns to be updated. The value1 and value2 parameters are the new values to be set. The condition parameter is the condition that must be met for a row to be updated.

Delete Table:

MySQL Connector/Python allows users to delete tables. To delete a table, the DROP TABLE statement is used. The syntax for this statement is as follows:

DROP TABLE table_name;

In this statement, the table_name parameter is the name of the table to be deleted.

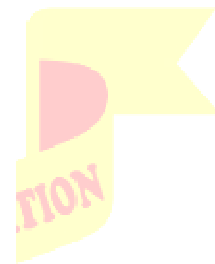
COMMITTED TO EXCELLENCE IN EDUCATION
निर्मलस्नेह उत्तम सेवाधर्म

Practical-4: Write a Python Program to work with databases in Python to perform operations such as

a. Connecting to database
b. Creating and dropping tables
c. Inserting and updating into tables.

Source code:

```
import mysql.connector as msc
#Connecting to database
mydb = msc.connect(user='noteve',
passwd='noteve', host='localhost',
database = 'mydb')

if mydb.is_connected():
     print("Successfully connected")

#creating cursor
mycursor = mydb.cursor()

#creating table
mycursor.execute("CREATE TABLE customers
(name VARCHAR(255), address VARCHAR(255))")

#Inserting data into table
sql = "INSERT INTO customers (name, address)
VALUES (%s, %s)"
val = ("Eve", "Highway 21")
mycursor.execute(sql, val)

#Commiting the changes
mydb.commit()

#Updating data in table
sql = "UPDATE customers SET address = %s
WHERE name = %s"
val = ("90 Feet Road", "Eve")
mycursor.execute(sql, val)

#Commiting the changes
mydb.commit()

#Dropping Table
mycursor.execute("DROP TABLE customers")
```



Outputs:

a)Connecting to database

```
Successfully connected
```

b)Creating and dropping tables

```
MariaDB [mydb]> show tables from mydb;
+----------------+
| Tables_in_mydb |
+----------------+
| customers      |
+----------------+
1 row in set (0.001 sec)
```

```
MariaDB [mydb]> show tables from mydb;
Empty set (0.000 sec)
```

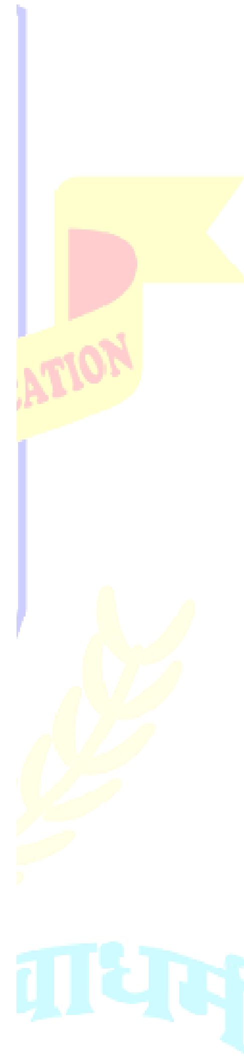
c) Inserting and updating into tables.

```
MariaDB [mydb]> select * from customers;
+------+------------+
| name | address    |
+------+------------+
| Eve  | Highway 21 |
+------+------------+
1 row in set (0.001 sec)

MariaDB [mydb]> 2023-03-24 22:28:33 43 [Warning] Aborted connectio
n 43 to db: 'mydb' user: 'noteve' host: 'localhost' (Got an error
reading communication packets)

MariaDB [mydb]> select * from customers;
+------+--------------+
| name | address      |
+------+--------------+
| Eve  | 90 Feet Road |
+------+--------------+
1 row in set (0.000 sec)
```

Example 1: Write a Python program to insert and delete a row in table.

Source code:

```
import mysql.connector as msc


mydb = msc.connect(user='noteve',
passwd='noteve', host='localhost',
database = 'mydb')

if mydb.is_connected():
     print("Successfully connected")

#creating cursor
mycursor = mydb.cursor()


#Inserting data into table
sql = "INSERT INTO students (name, address)
VALUES (%s, %s)"
val = ("Aafraaz", "Asalpha 90 Feet Road")
val1 = ("Ashfak", "Kurla Street 12")
val2 = ("Meraj", "Mahim Avenue 11")
val3 = ("Eve", "Highway 21")

mycursor.execute(sql, val)
mycursor.execute(sql, val1)
mycursor.execute(sql, val2)


#Commiting the changes
mydb.commit()

#delete a row from table
sql = "delete from students where name =
'Eve' "
mycursor.execute(sql)

#Commiting the changes
mydb.commit()

mycursor.close()
mydb.close()
```



Output:

```
Successfully connected
```

```
MariaDB [mydb]> select * from students;
+---------+----------------------+
| name    | address              |
+---------+----------------------+
| Aafraaz | Asalpha 90 Feet Road |
| Ashfak  | Kurla Street 12      |
| Meraj   | Mahim Avenue 11      |
| Eve     | Highway 21           |
+---------+----------------------+
4 rows in set (0.000 sec)

MariaDB [mydb]> select * from students;
+---------+----------------------+
| name    | address              |
+---------+----------------------+
| Aafraaz | Asalpha 90 Feet Road |
| Ashfak  | Kurla Street 12      |
| Meraj   | Mahim Avenue 11      |
+---------+----------------------+
3 rows in set (0.001 sec)
```

Example 2: Write a Python program to create a database table.

Source code:

```
#importing mysql.connector
import mysql.connector

#creating connection
mydb = mysql.connector.connect(
      host="localhost",
      user="noteve",
      passwd="noteve"
 )

#creating cursor
mycursor = mydb.cursor()

#Creating Database
mycursor.execute("CREATE DATABASE
mydatabase")

#creating table
mycursor.execute("CREATE TABLE Students
(name VARCHAR(255), address VARCHAR(255))")

mycursor.close()
mydb.close()
```

Output:

```
MariaDB [mydatabase]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydatabase         |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
7 rows in set (0.002 sec)

MariaDB [mydatabase]> show tables from mydatabase;
+----------------------+
| Tables_in_mydatabase |
+----------------------+
| Students             |
+----------------------+
1 row in set (0.002 sec)
```

Example 3: Write a Python program for exception handling in database

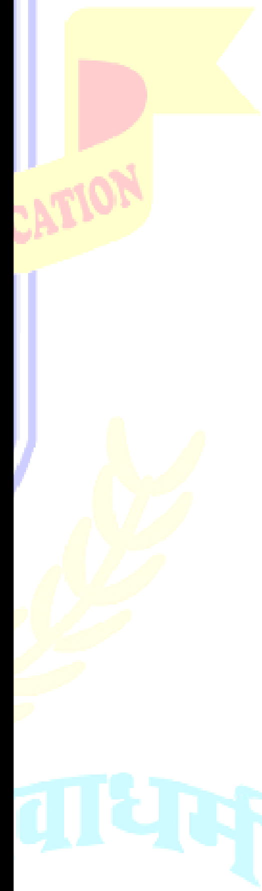
Source code:

```
import mysql.connector
#creating connection
try:
    mydb = mysql.connector.connect(
         host="localhost",
         user="noteve",
         passwd="noteve"
     )

except mysql.connector.Error as err:
    print("Error in connection", err)

else:
    print("Connection successful")

#creating cursor
try:
    mycursor = mydb.cursor()
except mysql.connector.Error as err:
    print("Error in creating cursor", err)
else:
    print("Cursor Created Successfully")

#Creating Database
try:
    mycursor.execute("CREATE DATABASE
DataBase1")
except mysql.connector.Error as err:
    print("Error in creating Database", err)
else:
    print("Database Created Successfully")

#creating table
try:
    mycursor.execute("USE DataBase1")
    mycursor.execute("CREATE TABLE Students
(name VARCHAR(255), address VARCHAR(255))")
except mysql.connector.Error as err:
    print("Error in creating table", err)
else:
    print("Table Created Successfully")


try:
    mycursor.close()
    mydb.close()
except mysql.connector.Error as err:
    print("Error in closing database
connection", err)
else:
    print("Database connection closed
successfully")
```

Output:

```
Connection successful
Cursor Created Successfully
Database Created Successfully
Table Created Successfully
Database connection closed successfully
```

```
MariaDB [mydatabase]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydatabase         |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
7 rows in set (0.002 sec)

MariaDB [mydatabase]> show databases;
+--------------------+
| Database           |
+--------------------+
| DataBase1          |
| information_schema |
| mydatabase         |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
8 rows in set (0.001 sec)

MariaDB [mydatabase]> show tables from DataBase1;
+---------------------+
| Tables_in_DataBase1 |
+---------------------+
| Students            |
+---------------------+
1 row in set (0.001 sec)
```



Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. How to connect to a Database with Python sql.connector module ?
2. How to Create a Database with Python sql.connector module ?
3. How to Create a Table with Python ?
4. How to Insert a row in a table with Python?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 5

Exception Handling

Exception handling is a process of dealing with errors that occur during the execution of a program. In Python, exceptions can be handled using try and except statements. The try block contains code that may throw an exception. The except block is used to handle the exception that is thrown.

The try statement is used to define a block of code to be tested for errors. The code within the try block is executed first, and if an exception occurs, the except clause is executed.

The syntax of the try statement is as follows:

```
try:
   # code to be executed
except:
   # code to be executed if an exception occurs
```

For example,

```
try:
   print(x)
except:
   print("An exception occurred")
```

In this example, the try block contains code that attempts to print the value of the variable x. If the variable does not exist, then an exception is thrown and the except clause is executed.

Using the try statement, errors can be caught and handled. This allows the program to continue running, even if an exception is encountered.

The except statement can be used to catch specific types of exceptions. This allows different exceptions to be handled differently. For example,

```
try:
   print(x)
except NameError:
   print("Variable x is not defined")
except:
   print("Something else went wrong")
```

In this example, the except statement has two clauses. The first clause catches NameError exceptions and prints an error message. The second clause catches all other exceptions and prints a different error message.

The try statement can also be used with an else clause. The else clause is executed if no exception is thrown. For example,

```
try:
   print(x)
except:
   print("Something went wrong")
else:
   print("Nothing went wrong")
```

The else clause is only executed if no exception is thrown.

Finally, the try statement can be used with a finally clause. The finally clause is executed no matter what. It is used to perform clean-up tasks, such as closing files or releasing resources. For example,

```
try:
   print(x)
except:
   print("Something went wrong")
finally:
   print("The 'try except' is finished")
```

The finally clause is executed regardless of whether or not an exception is thrown.

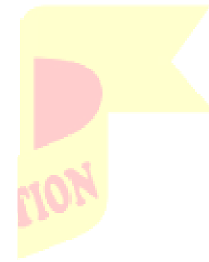
Exception handling is an important part of writing robust and reliable code. It allows errors to be caught and handled gracefully, allowing the program to continue running. The try, except, else, and finally statements can be used to handle exceptions in Python.

COMMITTED TO EXCELLENCE IN EDUCATION
निर्मलस्नेह उत्तम सेवाधर्म

Practical-5: Write a Python Program to demonstrate different types of exception handing.

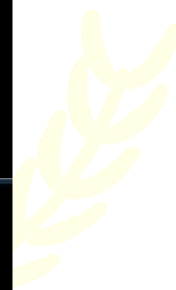
Source code:

```
try:
    numerator = int(input("Enter a number to
divide: "))
    denominator = int(input("Enter a number
to divide by: "))
    result = numerator / denominator

except ZeroDivisionError as e:
    print("You can't divide by zero!
idiot!")

except ValueError as e:

    print("Enter only numbers plz")

else:
    print(result)

finally:
    print("finally block:This will always
execute")
```



Output:

```
Enter a number to divide: 4
Enter a number to divide by: 2
2.0
finally block:This will always execute
```



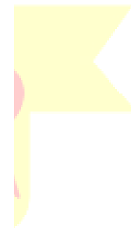
```
Enter a number to divide: 2
Enter a number to divide by: 0
You can't divide by zero! idiot!
finally block:This will always execute
```

```
Enter a number to divide: 2
Enter a number to divide by: A
Enter only numbers plz
finally block:This will always execute
```

Example-1: Write a python program to demonstrate exception handling from 'myfile.txt'.

Source code:

```
# Create a file
try:
    f = open("myfile.txt", "w+")
    if f.closed:
        print("File is closed")
    else:
        print("File is open")
    # Write file
    f.write("This is noteve")
except:
    print("Something went wrong when writing
the file")
else:
    print("File write was a success")
finally:
    f.close()

# Read file
try:
    f = open("myfile.txt", "r")
    print(f.read())
except:
    print("Something went wrong when reading
the file")
else:
    print("File read was a success")
finally:
    f.close()
    print("File is closed")
```



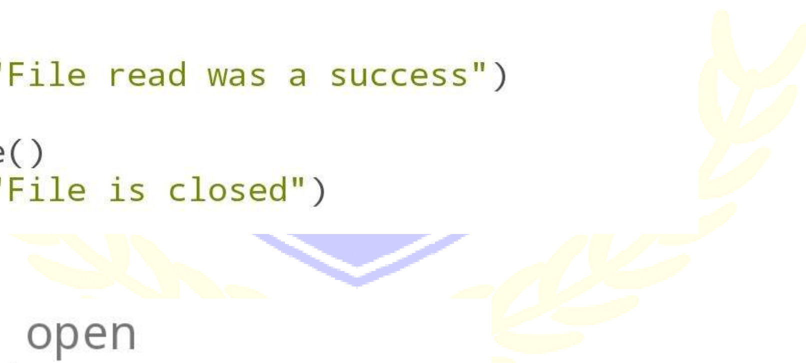
Output:

```
File is open
File write was a success
This is noteve
File read was a success
File is closed
```

Example-2: Write a python program to demonstrate exception handling from 'sample.txt'.

Source code:

```
# Create a file
try:
    f = open("sample.txt", "w+")
    if f.closed:
        print("File is closed")
    else:
        print("File is open")
    # Write file
    f.write("This is noteve")
except:
    print("Something went wrong when writing
the file")
else:
    print("File write was a success")
finally:
    f.close()

# Read file
try:
    f = open("sample.txt", "r")
    print(f.read())
except:
    print("Something went wrong when reading
the file")
else:
    print("File read was a success")
finally:
    f.close()
    print("File is closed")
```



Output:

```
File is open
File write was a success
This is noteve
File read was a success
File is closed
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What is Exception Handling?
2. When does except block runs?
3. When does else block runs?
4. When does finally block runs?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 6

Graphical User Interface

Graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices such as computers, smartphones, and tablets through graphical icons and visual indicators such as secondary notation, instead of text-based command line interfaces, typed command labels or text navigation.

Python is a great language when it comes to creating graphical user interfaces (GUIs). It provides several options for developing graphical user interfaces, such as Tkinter, wxPython, PyQt, Kivy, and PyGObject. Tkinter is the most commonly used and the most basic GUI framework available in Python.

Tkinter is an inbuilt Python module used to create simple GUI applications. It is the most commonly used module for designing Graphical User Interfaces (GUIs) with Python. Tkinter is a powerful toolkit that provides a variety of controls, such as buttons, labels, text boxes, checkboxes, and more, to create a graphical user interface for desktop applications.

Tkinter is easy to use and understand. It provides a platform-independent interface to the Tk GUI toolkit, which is available for most operating systems. It is also used for rapid application development as it provides a set of standard widgets.

The basic components of a Tkinter GUI application are the root window, frames, widgets, and geometry management. The root window is the main window in which all other components are placed. The frames are used to divide the root window into sections. Widgets are the graphical elements that allow a user to interact with the GUI application. Geometry management is used to organize and arrange the widgets within the frames.

To create a GUI application using Tkinter, one must first import the Tkinter module. To create the main window, the Tk() constructor is used. After creating the main window, one can create frames and widgets, and organize them using geometry management.

To create buttons, labels, text boxes, and other widgets, the widget classes must be imported from the Tkinter module. To create a simple button, the Button() class is used. For example, to create a button with the text “Click Me”, the following code can be used:

```
btn = Button(root, text="Click Me")
```

The widget classes also provide options to configure the appearance of the widgets. For example, to change the background color of a button, the bg option can be used.

The Tkinter module also provides support for creating menu bars and toolbars. To create a menu bar, the Menu() class is used. To create a toolbar, the Toolbar() class is used.

Practical-6 : Write a GUI Program in Python to design application that demonstrates

a. Different fonts and colors
b. Different Layout Managers
c. Event Handling

Source code:

```
from tkinter import *

root = Tk()

root.title("Fonts, Colors and Layout
Managers Demo")

font_label = Label(root, text="Different
fonts example", font=("Verdana", 20,
"bold"))

color_label = Label(root, text="Different
colors example", bg="pink", fg="blue")

layout_label = Label(root, text="Different
layout managers example")

def handle_event(event):
    layout_label.config(text="Button
clicked")

button1 = Button(root, text="Click me")
button1.bind("<Button-1>", handle_event)

font_label.grid(row=0, column=0)
color_label.grid(row=1, column=0)
layout_label.grid(row=2, column=0)
button1.grid(row=3, column=0)

root.mainloop()
```

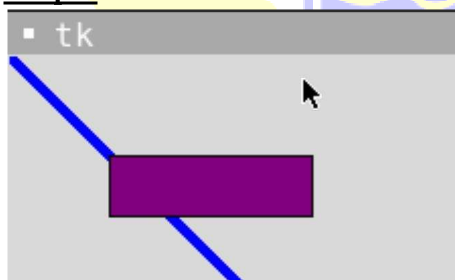
Output:

Example-1: Write a python program to create line with rectangle having canvas width =200 and height =100.

Source code:

```
from tkinter import *
window = Tk()
canvas = Canvas(window,height=100,
 width = 200)
canvas.create_line(0,0,50,50,fill='blue',
width =5)
canvas.pack()
canvas.create_rectangle(20,20,30,30, fill=
"purple")
canvas.pack()
window.mainloop()
```

Output:

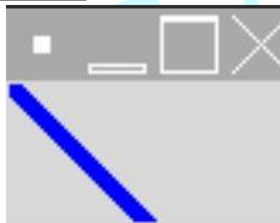


Example-2: Write python program for creating line with canvas width=80 and height=40.

Source code:

```
from tkinter import *
window = Tk()
canvas = Canvas(window,height=40,
 width = 80)
canvas.create_line(0,0,500,500,fill='blue',
width =5)
canvas.pack()
window.mainloop()
```

Output:



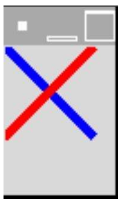
EXCELLENCE IN EDUCATION
स्नेह उत्तम सेवाधर्म

Example-3: write a python program for creating two lines with canvas width 60 and height 80.

Source code:

```
from tkinter import *
window = Tk()
canvas = Canvas(window,height=80,
 width = 60)
canvas.create_line(0,0,50,50,fill='blue',
width =5)
canvas.pack()
canvas.create_line(0,50,50,0, fill= "red",
width = 5)
canvas.pack()
window.mainloop()
```

Output:

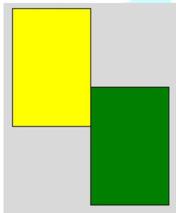


Example-4: write a python program to draw 2 rectangles of yellow and green colors.

Source code:

```
from tkinter import *
window = Tk()
canvas = Canvas(window,height=500,
 width = 500)
canvas.create_rectangle(50,50,150,200,fill='
yellow')
canvas.pack()
canvas.create_rectangle(150,150,250,300,
fill= "green")
canvas.pack()
window.mainloop()
```

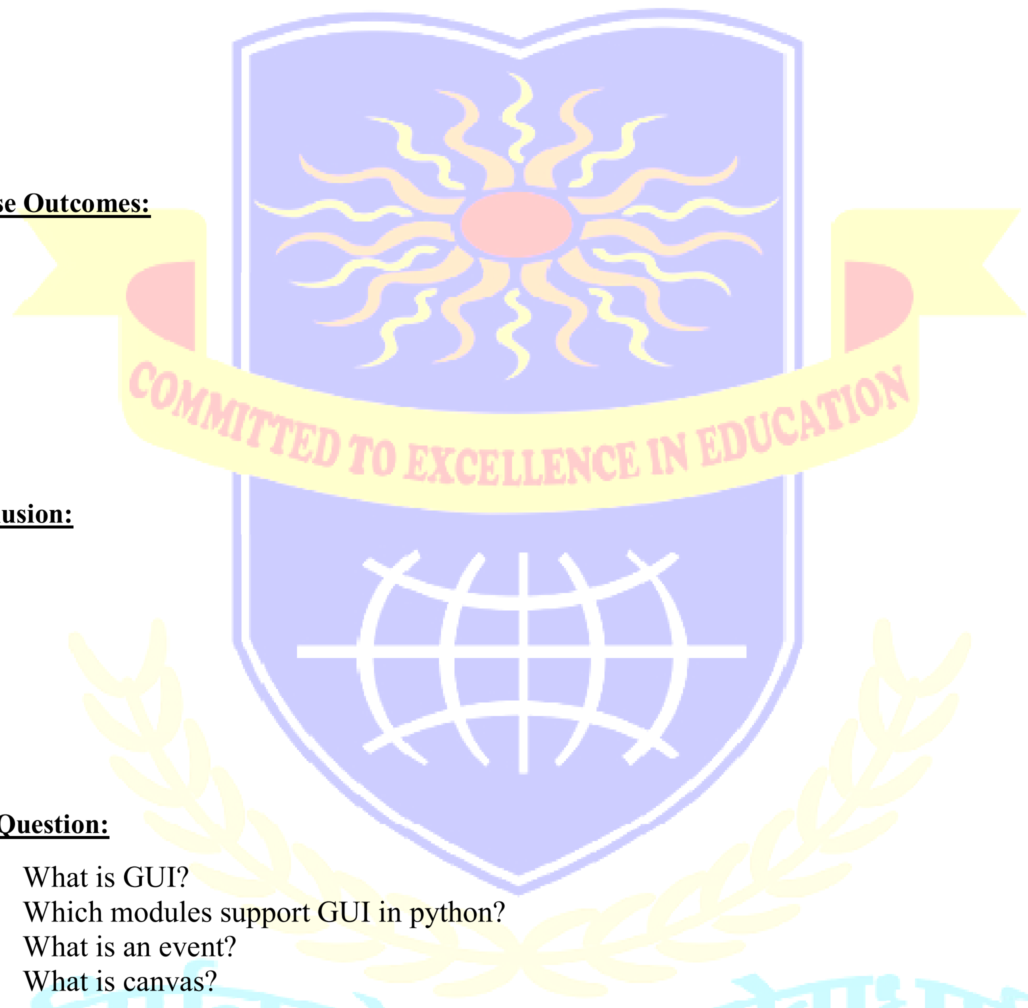
Output:



COMMITTED TO EXCELLENCE IN EDUCATION

Result and Discussion:

Learning Outcomes:



Course Outcomes:

Conclusion:

Viva Question:

1. What is GUI?
2. Which modules support GUI in python?
3. What is an event?
4. What is canvas?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 7
Date and Time in Python

The Python date and time module is a powerful tool for managing dates, times, and timestamps. It is a valuable resource for developers and provides many features that make working with dates and times easier.

The Python date and time module is part of the standard library, so it does not need to be installed separately. It provides a set of classes and functions for manipulating, formatting, and converting dates and times. It can be used to convert strings to dates and times, to compare dates, to add and subtract time from a given date, and to format dates and times into strings.

The Python date and time module provides several classes for working with dates and times. The most commonly used classes are the date, time, datetime, and timedelta classes. The date class handles dates, the time class handles times, the datetime class handles date and time, and the timedelta class handles differences in time.

The date class is the simplest of the four classes and can be used to store dates in the form of year, month, and day. The time class is used to store time in the form of hour, minute, second, and microsecond. The datetime class is used to store both date and time, and the timedelta class is used to store differences in time.

The date and time module also provides several functions that can be used to manipulate and format dates and times. The strftime() function can be used to format dates and times into strings. The strptime() function can be used to convert strings to dates and times. The strftime() and strptime() functions allow developers to specify the format of the date and time strings and to convert the strings to and from different formats.

The date and time module also provides several functions for comparing dates and times. The date and time comparison functions allow developers to compare dates and times and to determine if a date or time is before or after another date or time. The date and time arithmetic functions allow developers to add and subtract time from a given date and time.

The Python date and time module also provides several functions for calculating the amount of time between two dates and times. The timedelta() function can be used to calculate the amount of time between two dates and times, and the total_seconds() function can be used to calculate the amount of time between two dates and times in seconds.

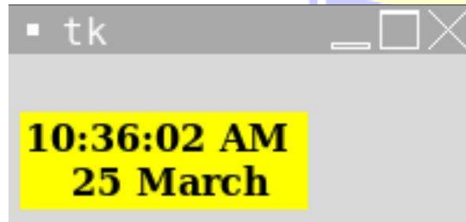
The Python date and time module is an invaluable tool for developers. It provides a set of classes and functions for managing dates and times, and it provides functions for formatting and converting dates and times. It also provides functions for comparing and calculating differences in dates and times. With the Python date and time module, developers can easily work with dates and times in their applications.

Practical-7: Write Python Program to create application, which uses date and time in Python.

Source code:

```
import tkinter  as tk
from tkinter import ttk
my_w = tk.Tk()
my_w.geometry("200x200")
from time import strftime
def my_time():
    time_string = strftime('%H:%M:%S %p \n
%d %B')
     # time format
    l1.config(text=time_string)
    l1.after(1000,my_time) # time delay of
1000 milliseconds

my_font=('times',12,'bold') # display size
and style

l1=tk.Label(my_w,font=my_font,bg='yellow')
l1.grid(row=1,column=1,padx=5,pady=25)

my_time()
my_w.mainloop()
```

Output:



Example 1: Write a Python Program for combining date and time.

Source code:

```
# Program to combine date and time
import datetime

date1 = datetime.date(year = 2020, month =
5, day = 5)
time1 = datetime.time(hour=14, minute=30,
second=45)
dateTime1 = datetime.datetime.combine(date1,
time1)

print("Combined date and time :", dateTime1)
```

Output:

```
Combined date and time : 2020-05-05
14:30:45
```

Example 2: Write a Python Program to find duration by using "Time Delta"

Source code:

```
from datetime import datetime, timedelta

date_format = "%Y-%m-%d %H:%M:%S"

date1 = datetime.strptime("2023-03-23
20:27:32", date_format)
date2 = datetime.strptime("2023-04-1
13:24:27", date_format)

duration = date2 - date1

print("Duration is:",duration)
```

Output:

```
Duration is: 8 days, 16:56:55
```

Example 3: Write a Python Program to compare two dates.

Source code:

```
import datetime

date1 = datetime.date(2023,3,23)
date2 = datetime.date(2023,4,2)

difference = date2 - date1

differenced = difference.days

print(differenced)

if differenced > 0:
    print("date2 is greater than date1")
else:
    print("date1 is greater than date2")
```

Output:

```
10
date2 is greater than date1
```

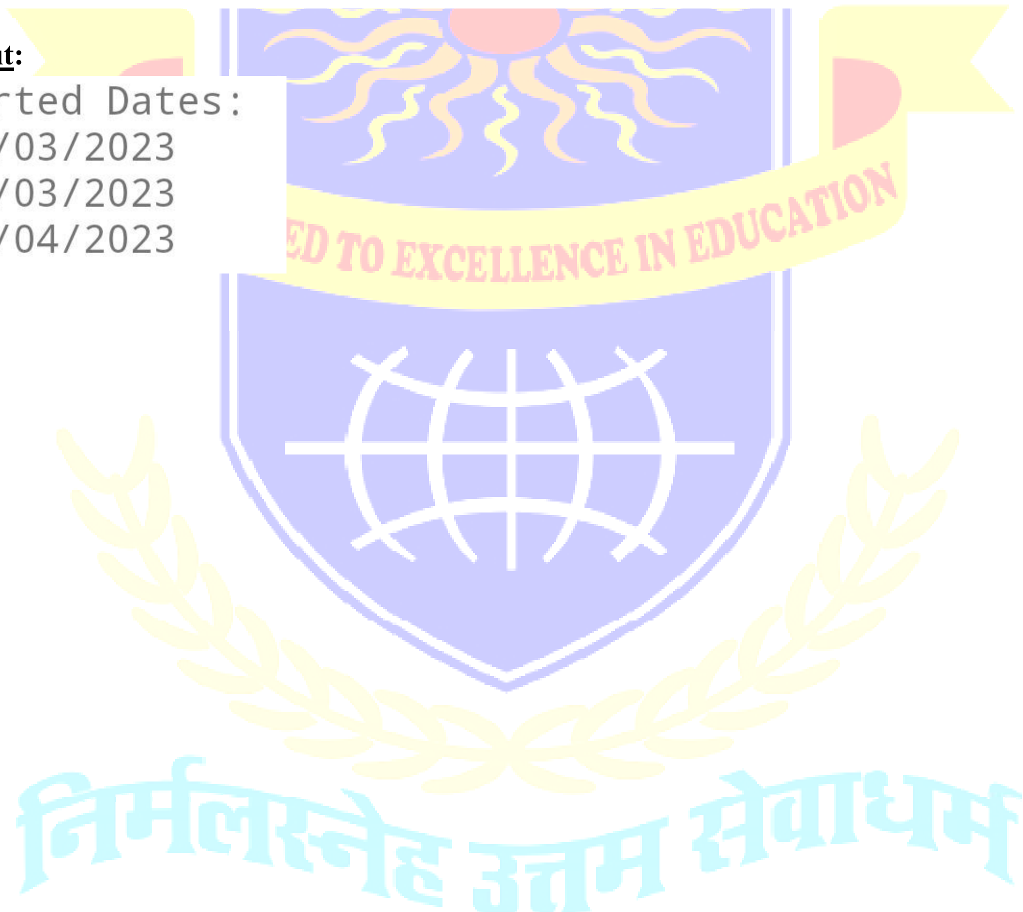
Example 4: Write a Python Program to sort the dates

Source code:

```
import datetime
date_str_list = ['11/03/2023', '23/03/2023',
'13/04/2023']

date_list = []
for date in date_str_list:
    date_list.append(datetime.datetime.strptime(date,
'%d/%m/%Y'))

date_list.sort()

print("Sorted Dates:")
for date in date_list:
    print(date.strftime('%d/%m/%Y'))
```

Output:

```
Sorted Dates:
11/03/2023
23/03/2023
13/04/2023
```



Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. Which module supports date and time?
2. Which method is used to print current date and time?
3. Which method is used to print current date?
4. Which method is used to print current time?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 8

Client Server information

The socket module can be used to create server applications that can handle multiple client connections efficiently.

This module allows programs to create sockets, bind them to specific ports and listen for incoming connections. It also provides methods for sending and receiving data over the network.

The socket module in Python provides a set of methods and classes for working with TCP/IP sockets.

Socket():

Socket() is used to create a socket object that can be used to establish a connection between two machines. It takes the address family, socket type, and protocol type as arguments and returns a socket object.

Bind():

Bind() is used to bind a socket to a specific network address. It takes the socket object and a tuple containing an IP address and port number as arguments and binds the socket to that address.

Accept():

Accept() is used to accept incoming connections. It takes a socket object as an argument and waits for an incoming connection. When a connection is accepted, it returns a new socket object that can be used to communicate with the connected client.

Encode():

Encode() is used to convert a string or a sequence of bytes into a unicode string. It takes a string as an argument and returns a unicode string.

Decode():

Decode() is used to convert a unicode string into a string or a sequence of bytes. It takes a unicode string as an argument and returns a string.

Send():

Send() is used to send data over a socket. It takes a socket object and a string or a sequence of bytes as arguments and sends the data over the socket.

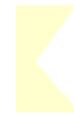
Close():

Close() is used to close a socket connection. It takes a socket object as an argument and closes the connection.

Practical-8 : Write a Python program to create server-client and exchange basic information

Source code:

```
#Server Code:

import socket

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Bind to the port
s.bind((host, port))

# Wait for one connection
s.listen(1)

print("Waiting for any incoming connections ...")

conn, addr = s.accept()

print(f"Received connection from {addr[0]}")

# Receive data from client
data = conn.recv(1024).decode()

print(f"Received data: {data}")


# Send data to client
data = "Thanks for connecting"
conn.send(data.encode())


# Close the connection
conn.close()
```



```python
#Client Code:

import socket

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Connect to server
s.connect((host, port))

# Send data to server
data = "Hello from Client!"
s.send(data.encode())

# Receive data from server
data = s.recv(1024).decode()

print(f"Received data: {data}")

# Close the connection
s.close()
```

Output:

```
~ $ python s3.py
Waiting for any incoming connections ...
```

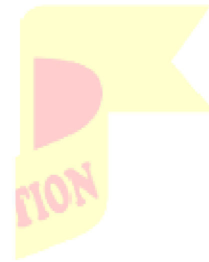
```
~ $ python s3.py
Waiting for any incoming connections ...
Received connection from 127.0.0.1
Received data: Hello from Client!
```

```
~ $ python c3.py
Received data: Thanks for connecting
~ $
```

Example-1: write a python program to create a server client and exchange current-day information.

Source code:

```
import socket
import datetime

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Bind to the port
s.bind((host, port))

# Wait for one connection
s.listen(1)

print("Waiting for any incoming connections ...")

conn, addr = s.accept()

print(f"Received connection from {addr[0]}")

# Receive data from client
data = conn.recv(1024).decode()

print(f"Received data: {data}")

# Send data to client
now = datetime.datetime.now()
data = "Today is " + now.strftime("%A %d %B %Y")
conn.send(data.encode())


# Close the connection
conn.close()
```



```
#Client Code:

import socket

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Connect to server
s.connect((host, port))

# Send data to server
data = "What day is it today?"
s.send(data.encode())

# Receive data from server
data = s.recv(1024).decode()

print(f"Received data: {data}")

# Close the connection
s.close()
```

Output:

```
~ $ python s4.py
Waiting for any incoming connections ...
Received connection from 127.0.0.1
Received data: What day is it today?

~ $ python c4.py
Received data: Today is Sunday 26 March 2023
~ $
```

Example-2: write a python program to create a server-client and exchange current time information.

Source code:

```
#Server Code:

import socket
import datetime

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Bind to the port
s.bind((host, port))

# Wait for one connection
s.listen(1)

print("Waiting for any incoming connections ...")

conn, addr = s.accept()

print(f"Received connection from {addr[0]}")

# Receive data from client
data = conn.recv(1024).decode()

print(f"Received data: {data}")

# Send data to client
now = datetime.datetime.now()
data = "The current time is " + now.strftime("%X")
conn.send(data.encode())


# Close the connection
conn.close()
```

```python
#Client Code:

import socket

# Create a socket object
s = socket.socket()

# Get local machine name
host = socket.gethostname()

# Reserve a port
port = 12345

# Connect to server
s.connect((host, port))

# Send data to server
data = "What time is it?"
s.send(data.encode())

# Receive data from server
data = s.recv(1024).decode()

print(f"Received data: {data}")

# Close the connection
s.close()
```

Output:

```
~ $ python s5.py
Waiting for any incoming connections ...
Received connection from 127.0.0.1
Received data: What time is it?
```

```
~ $ python c5.py
Received data: The current time is 10:48:32
~ $
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What is a socket?
2. What is TCP/IP?
3. What does listen method do?
4. What does accept method do?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 9

Implementation of OOP concept

Object-Oriented Programming (OOP) is an approach to programming which focuses on objects and their interactions to solve problems. OOP is based on the concept of objects, which are collections of related data and methods that work with that data. Objects can be thought of as the physical representation of a real-world object, such as a car, a book, or a person.

In OOP, programs are organized around objects, which are collections of data and related methods. Objects encapsulate data, meaning that all of the data associated with an object is contained within the object itself. This makes it easier to maintain and update programs, since all the related data is packaged together. Objects also have methods, which are functions that manipulate the data within the object.

When programming with OOP, objects are created from classes, which are templates that define the data and methods associated with the object. Classes are like blueprints, and when you create an object from a class, you are creating an instance of the class. Objects created from the same class will have similar data and methods, but the data can be different for each instance of the class.

OOP also makes use of inheritance, which allows one class to inherit the data and methods of another class. This makes it easier to create objects that share a lot of the same functionality. Inheritance also makes it easier to update and maintain code, since changes only need to be made to the parent class, and all of the child classes will inherit the changes.

Polymorphism is a core principle of OOP that allows objects of different classes to be treated the same way. This means that objects can respond differently when the same methods are called on them, depending on the object's class.

Encapsulation is a principle of OOP that helps to protect data within an object. It prevents external code from directly manipulating an object's data, and instead requires that access to the data be done through the object's methods. This helps to ensure that objects maintain a consistent internal state and that any changes to the data are done in a controlled and consistent manner.

Abstraction is a principle of OOP that allows programmers to hide the details of how an object works and instead focus on how the object is used. Abstraction is used to simplify complex tasks and make code more readable, and it is also used to reduce code duplication by removing common code from multiple objects and placing it in a single location.

OOP is a powerful approach to programming that makes it easier to maintain and update programs.

Practical-9 : Write a Python program to implement concepts of OOP such as Types of Methods, Inheritance, Polymorphism.

Source code:

```
class Students:
  def __init__(self, name, rollNo):
    self.name = name
    self.rollNo = rollNo
#Types of Methods

#Instance Method
  def show_name(self):
    print("Name : {}".format(self.name))

  def show_rollNo(self):
    print(f"Roll No: {self.rollNo}")


#Class Method
  @classmethod
  def show_type(cls):
    print("This is a class method")

#Static Method
  @staticmethod
  def show_static():
    print("This is a static method")

#Inheritance

class CS(Students):
  #Polymorphism
  def show_name(self):
    print(f"Student's Name:{self.name}")

student1 = Students("Saad", "2")
student1.show_name()
student1.show_rollNo()

s1 = CS("Saud", "1")
s1.show_name()
s1.show_rollNo()


Students.show_type()
CS.show_static()
```

Output:

```
Name : Saad
Roll No: 2
Student's Name:Saud
Roll No: 1
This is a class method
This is a static method
```

Example 1: Write a Python program to implement the concept of Inheritance with one base class and two child classes.

Source code:

```
class Students:
    def __init__(self,name,rollNo):
        self.name = name
        self.rollNo = rollNo

    def students_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollNo}\n")

class FYCs(Students):
    pass

class SYCs(Students):
    pass

s1 = FYCs("Saad",1)
s2 = SYCs("Saud",2)

s1.students_details()
s2.students_details()
```

Output:

```
Name: Saad
Roll No: 1

Name: Saud
Roll No: 2
```

Example 2: Write a Python program to implement multilevel inheritance.

Source code:

```
class Students:
    def __init__(self,name,rollNo):
        self.name = name
        self.rollNo = rollNo

    def students_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollNo}\n")

class FYCs(Students):
    pass

class SYCs(FYCs):
    pass

s1 = FYCs("Saad",1)
s2 = SYCs("Saud",2)

s1.students_details()
s2.students_details()
```

Output:

```
Name: Saad
Roll No: 1

Name: Saud
Roll No: 2
```

Example 3: Write a Python program to implement polymorphism: Define methods in the child class that have the same name as the methods in the parent class.

Source code:

```
class Students:
    def __init__(self,name,rollNo):
        self.name = name
        self.rollNo = rollNo

    def students_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollNo}\n")

class FYCs(Students):
    def students_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollNo}")
        print("Class: FYCS")



s1 = Students("Saad",1)
s2 = FYCs("Saud",2)

s1.students_details()
s2.students_details()
```

Output:

```
Name: Saad
Roll No: 1

Name: Saud
Roll No: 2
Class: FYCS
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What is a class?
2. What is an object?
3. What is Inheritance?
4. What is Polymorphism?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]

Theory 10
Implementation of OOP concept

Object-Oriented Programming (OOP) is an approach to programming which focuses on objects and their interactions to solve problems. OOP is based on the concept of objects, which are collections of related data and methods that work with that data. Objects can be thought of as the physical representation of a real-world object, such as a car, a book, or a person.

In OOP, programs are organized around objects, which are collections of data and related methods. Objects encapsulate data, meaning that all of the data associated with an object is contained within the object itself. This makes it easier to maintain and update programs, since all the related data is packaged together. Objects also have methods, which are functions that manipulate the data within the object.

An abstract method is a method that is declared, but not defined in a class. It is up to subclasses to provide the concrete implementation of the method. The purpose of an abstract method is to provide a common interface across all subclasses. It allows subclasses to provide their own implementation of the method, while still offering the same interface for other classes to use.

Abstract classes, on the other hand, are classes that cannot be instantiated. They contain one or more abstract methods, but also can contain concrete methods. An abstract class is used to provide a common definition of a base class that multiple subclasses can inherit from.

In Python, an interface is a way of defining a contract between classes to ensure that a certain set of methods are implemented by all classes that implement the interface. This can be useful for various types of applications, such as when writing a library of code that depends on a certain set of methods being implemented by the classes that use it. By defining an interface, a library can ensure that the code it uses will always have the same set of methods available, regardless of what class implements it. This can help ensure that code written using the library will always work, regardless of how the classes that implement the interface change over time.

Interfaces are also useful for ensuring that certain classes are used in particular ways. For example, if a particular class is intended to be used as a data structure, an interface can be used to ensure that the class implements all the methods necessary to use it as a data structure. This can help prevent code that uses the class from using it in a way that isn't intended.

In conclusion, abstract methods and classes, as well as interfaces, are all useful tools for ensuring that code is written in a way that is maintainable and consistent. Abstract methods and classes allow for code reuse, while interfaces help ensure that classes are used in the way they were intended. By using these tools, code can be written in a way that is both maintainable and readable.

Practical-10: Write a program to Python program to implement concepts of OOP such Abstract methods and classes,Interfaces

Source code:

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    #Interface
    @abstractmethod
    def go(self):
        pass

    @abstractmethod
    def stop(self):
        pass

class Car(Vehicle):

    def go(self):
        print("You drive the car")

    def stop(self):
        print("This car is stopped")

class Motorcycle:
    #Abstract class
    @abstractmethod
    def go(self):
        pass

    def stop(self):
        print("This Motorcycle is stopped")


class Bike(Motorcycle):
    def go(self):
        print("You ride the Bike")

    def stop(self):
        print("This Bike is stopped")


car = Car()
bike = Bike()

car.go()
bike.go()

car.stop()
bike.stop()
```

Output:

```
You drive the car
You ride the Bike
This car is stopped
This Bike is stopped
```

Example 1: Write a program to create a abstract class with name 'Bike' and having one method run().

Source code:

```
from abc import ABC,abstractmethod

class Bike(ABC):
    #Abstract class
    @abstractmethod
    def run(self):
        pass


class Honda(Bike):
    def run(self):
        print("This Honda bike is running")

bike = Honda()

bike.run()
```

Output:

```
This Honda bike is running
```

Example 2: Create the instance of Rectangle class, draw() method of rectangle class will be invoked.

Source code:

```
class Rectangle:
    def draw(self):
        print("Drawing Rectangle")

    def __init__(self):
        self.draw()


rec = Rectangle()
```

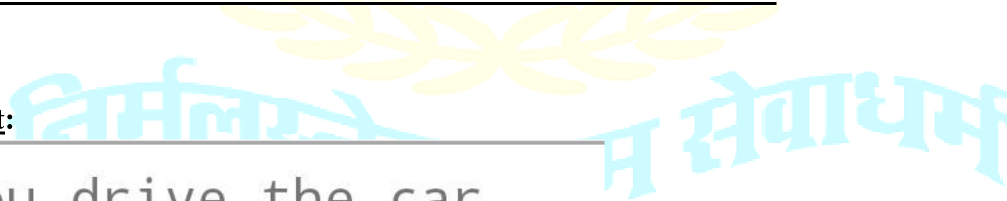
Output:

```
Drawing Rectangle
```

Example 3: Write a Python program to create interface having two abstract method and one subclass

Source code:

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    #Interface
    @abstractmethod
    def go(self):
        pass

    @abstractmethod
    def stop(self):
        pass

class Car(Vehicle):

    def go(self):
        print("You drive the car")

    def stop(self):
        print("This car is stopped")


car = Car()

car.go()

car.stop()
```



Output:

```
You drive the car
This car is stopped
```

Result and Discussion:

Learning Outcomes:

Course Outcomes:

Conclusion:

Viva Question:

1. What is an abstract class?
2. What is an abstract method?
3. What is an interface?
4. How to make an abstract class?

For Faculty Use

Correction Parameters	Formative Assessment [ ]	Timely completion of practical [ ]	Attendance Learning Attitude[ ]