MASTER OF COMPUTER APPLICATIONS (MCA)

Post Graduate Department of Computer Science and Technology

Sardar Patel University

Vallabh Vidyanagar, Gujarat, INDIA

PS01CMCA51 [Python Programming]

Unit-IV

MR. BHARAT B. PATEL (ASSOCIATE PROFESSOR)

(✉) bb_patel@spuvvn.edu

File Handling

- Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.
- The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short.
- Python treats file differently as text or binary and this is important.
- Each line of code includes a sequence of characters and they form text file.
- Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun.

File Handling

- **Python provides inbuilt functions for creating, writing and reading files.**
- **There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s and 1s).**

Text files

- In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.

Binary files

- In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

Steps to follow while working with file

Open the file

- **open()** function

Processing

- *Reading*
  - Using **read()** or **readline()** or **readlines()** function
- *Writing*
  - Using **write()** or **writelines()** function
- *Appending*
  - Using **append()** function

Close the file.

- Using **close()** function

Opening a file

- Before performing any operation on the file like read or write, first we have to open that file.
- For this, we should use Python's inbuilt function open().
- But at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

```
f = open(filename, mode)
```

Where the following mode is supported:

r : open an existing file for a read operation.

w : open an existing file for a write operation. If the file already contains some data then it will be overridden.

a : open an existing file for append operation. It won't override existing data.

r+ : To read and write data into the file. The previous data in the file will not be deleted.

w+ : To write and read data. It will override existing data.

a+ : To append and read data from the file. It won't override existing data.

File Access Modes

- Access modes govern the type of operations possible in the opened file.
- It refers to how the file will be used once its opened.
- These modes also define the location of the File Handle in the file.
- File handle is like a cursor, which defines from where the data has to be read or written in the file.
- There are 6 access modes in python.

File Access Modes

Read Only ('r')

- Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.

Write Only ('w')

- Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists.

Append Only ('a')

- Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

File Access Modes

Read and Write ('r+')

- Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.

Write and Read ('w+')

- Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.

Append and Read ('a+')

- Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Reading data from file

- **There are three ways to read data from a text file.**

Using read()

- Returns the read bytes in form of a string.
- Reads n bytes, if no n specified, reads the entire file.
- ***File_object.read([n])***

Using readline()

- Reads a line of the file and returns in form of a string.
- For specified n, reads at most n bytes.
- However, does not reads more than one line, even if n exceeds the length of the line.
- ***File_object.readline([n])***

Using readlines()

- Reads all the lines and return them as each line a string element in a list.
- *File_object.readlines()*

Writing data from file

- **There are two ways to write in a file.**

Using write()	Using writelines()
• Inserts the string str1 in a single line in the text file. • ***File_object.write(str1)***	• For a list of string elements, each string is inserted in the text file. • Used to insert multiple strings at a single time. • ***File_object.writelines(L)*** • ***For L = [str1, str2, str3]***

Python Module

- A module can contain executable statements as well as function definitions.
- A Python module is a file containing Python definitions and statements.
- A module can define functions, classes, and variables.
- A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.
- It also makes the code logically organized.

Import Module in Python – Import statement

- We can import the functions, classes defined in a module to another module using the import statement in some other Python source file.
- Syntax :

import module

Python Module

The from import Statement

- **Python's from statement lets you import specific attributes from a module without importing the module as a whole.**
- **Syntax :**

from module import * | attr1 , attr2 , …

GUI in Python

- Modern computer applications are user-friendly.
- User interaction is not restricted to console-based I/O.
- These applications can receive inputs through mouse clicks and can enable the user to choose from alternatives with the help of radio buttons, dropdown lists, and other GUI elements (or widgets).
- Python offers multiple options for developing GUI (Graphical User Interface).

PyQt	PyGTK	WxPython
It is, the Python interface to Qt, is a very popular cross-platform GUI framework.	It is the module that ports Python to another popular GUI widget toolkit called GTK.	Python wrapper around WxWidgets, another cross-platform graphics library.

Tkinter - GUI Toolkit for Python

- *Tkinter* is the standard GUI toolkit for Python developed by *Fredrik Lundh*.
- This module is bundled with standard distributions of Python for all platforms.
- Out of all the GUI methods, tkinter is the most commonly used method.
- It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications
- To create a tkinter app:

Importing the module – tkinter	Create the main window (container)	Add any number of widgets to the main window	Apply the event Trigger on the widgets.

GUI in Python

- **There are two main methods used which the user needs to remember while creating the Python application with GUI.**

Tk(screenName=None, baseName=None, className='Tk', useTk=1):

- To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'.
- To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:
- ***m=tkinter.Tk() where m is the name of the main window object.***

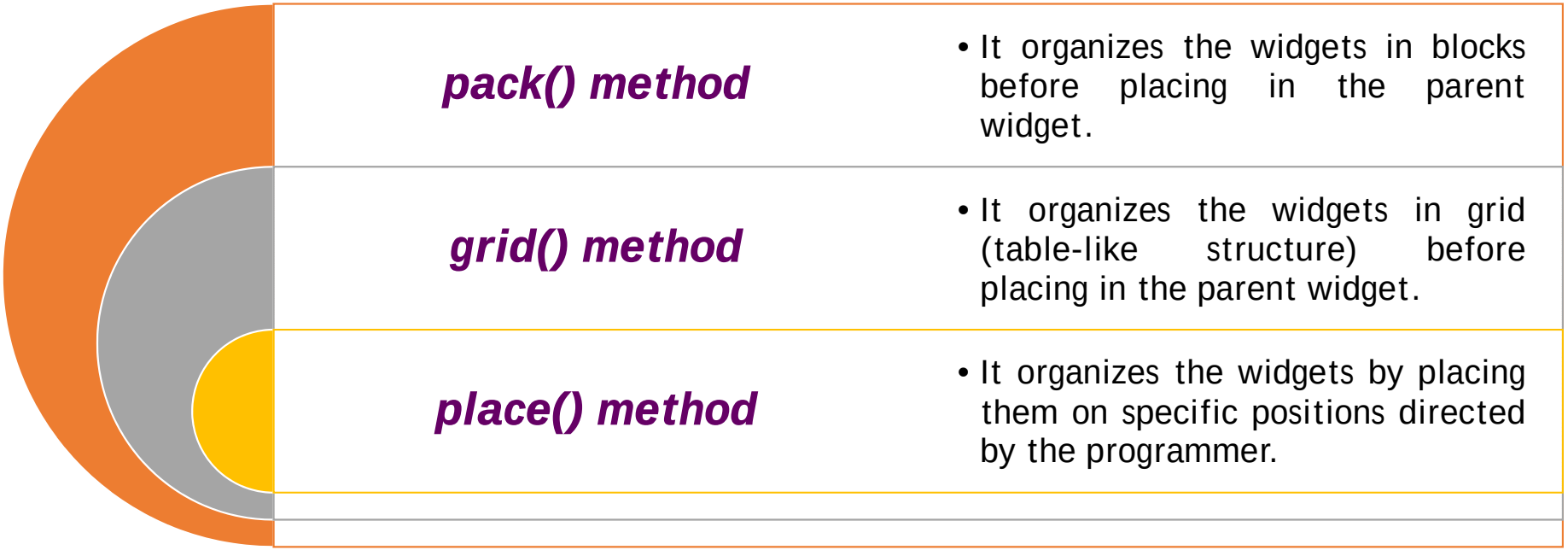
mainloop()

- There is a method known by the name mainloop() is used when your application is ready to run.
- mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
import tkinter
m = tkinter.Tk()
'''
widgets are added here
'''
m.mainloop()
```

GUI in Python

- **tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.**



pack() method	• It organizes the widgets in blocks before placing in the parent widget.
grid() method	• It organizes the widgets in grid (table-like structure) before placing in the parent widget.
place() method	• It organizes the widgets by placing them on specific positions directed by the programmer.

GUI in Python

- **There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.**

Button	Canvas	CheckButton	Entry	Frame	Label
Listbox	MenuButton	Menu	Message	RadioButton	Scale
Scrollbar	Text	TopLevel	SpinBox	PannedWindow	

Sample Program to Display Empty Window

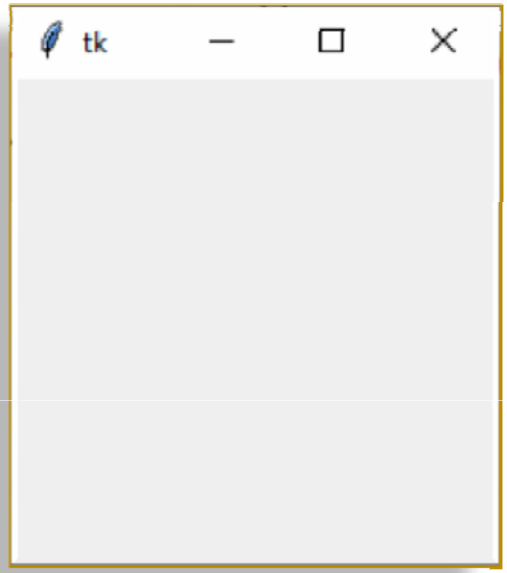
Code

GUI1.py - C:/Users/acer/Desktop/temp/GUI1.py (3.9.1)

File Edit Format Run Options Window Help

```
# Code to display window
import tkinter
top = tkinter.Tk ()
# Code to add widgets will go here...
top.mainloop ()
```

Output



Database Programming in Python

- The Python programming language has powerful features for database programming.
- Python supports various databases like MySQL, Oracle, Sybase, PostgreSQL, etc.
- Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements.
- For database programming, the Python DB API is a widely used module that provides a database application programming interface.

SQLite Database

- Python has built-in support for SQLite in the form of the sqlite3 module.
- SQLite is a self-contained transactional relational database engine that doesn't require a server configuration, as in the case of Oracle, MySQL, etc. It is an open source and in-process library developed by D. Richard Hipp in August 2000.
- The entire SQLite database is contained in a single file, which can be put anywhere in the computer's file system.
- SQLite is widely used as an embedded database in mobile devices, web browsers and other stand-alone applications.
- In spite of being small in size, it is a fully ACID compliant database conforming to ANSI SQL standards.

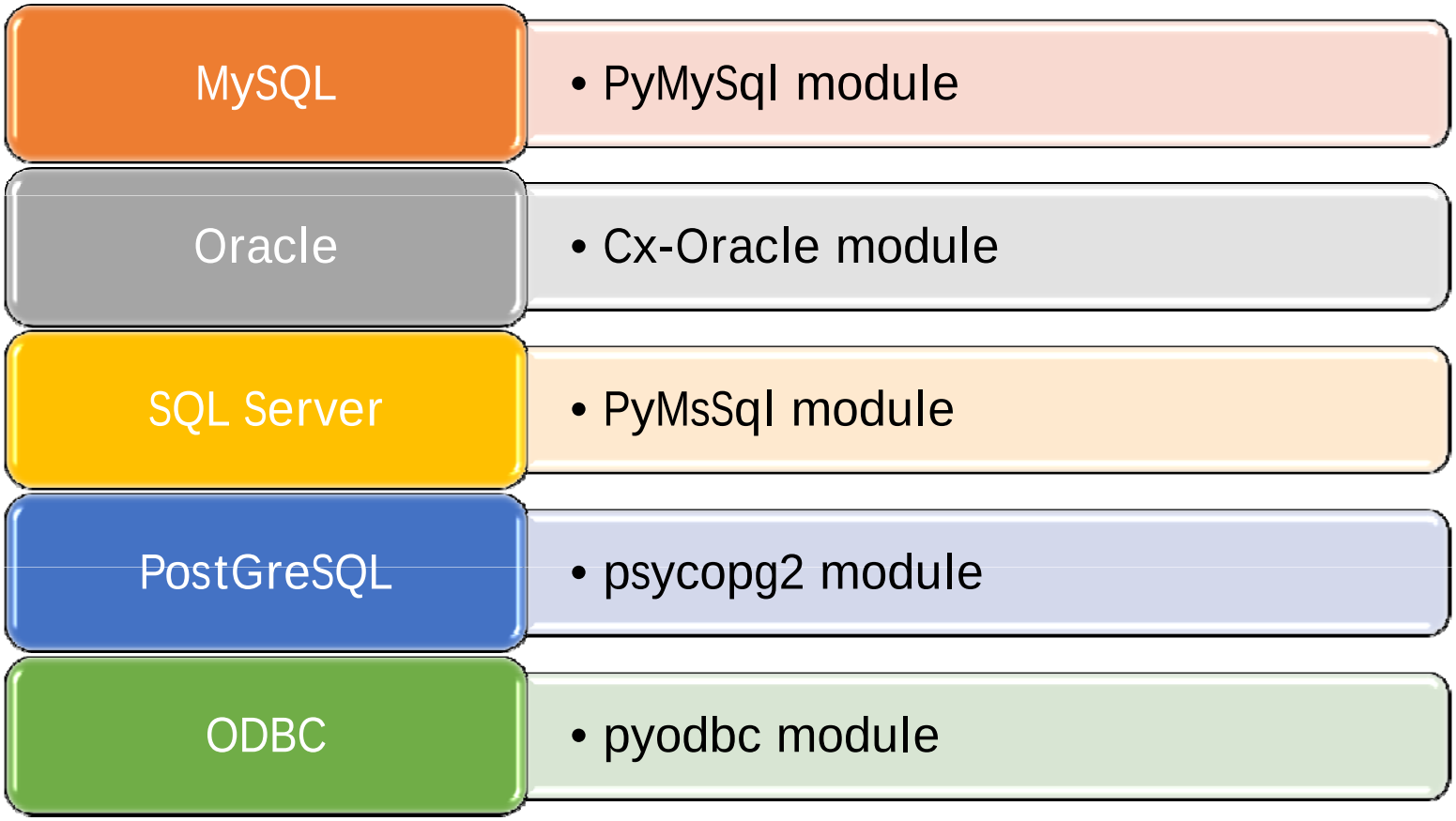
Benefits of Database Programming

- Programming in Python is arguably more efficient and faster compared to other languages.
- Python is famous for its portability.
- It is platform independent.
- Python supports SQL cursors.
- In many programming languages, the application developer needs to take care of the open and closed connections of the database, to avoid further exceptions and errors. In Python, these connections are taken care of.
- Python supports relational database systems.
- Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

DB-API (SQL-API) for Python

- Python Database API is a set of standards recommended by a Special Interest Group for database module standardization.
- Python modules that provide database interfacing functionality with all major database products are required to adhere to this standard.
- Standard Python distribution has in-built support for SQLite database connectivity. It contains sqlite3 module which adheres to DB-API 2.0 and is written by Gerhard Haring.
- Other RDBMS products also have DB-API compliant modules:

DB-API (SQL-API) for Python



Using Sqlite3 database

- **Python program to create a database (BBP.DB) and a table (STUD)**

```
import sqlite3

db=sqlite3.connect ( "BBP.db")
cur = db.cursor ()

cmd = """CREATE TABLE stud
          (
               pid     text,
               snm     text,
               per      int
          ) """
cur.execute (cmd)
print ( "Table is created in BBP.db")
db.close ()
```

Output

```
>>>
================== RESTART: C:\Us
Table is created in BBP.db
>>>
```

Using MySql database

- **Python MySQL Connector is a Python driver that helps to integrate Python and MySQL.**
- **This Python MySQL library allows the conversion between Python and MySQL data types.**
- **MySQL Connector API is implemented using pure Python and does not require any third-party library.**

Installation

- **Type the following *pip* command from Python terminal.**

pip install mysql-connector-python

Using MySql database

- Python MySQL Connector is a Python driver that helps to integrate Python and MySQL.
- This Python MySQL library allows the conversion between Python and MySQL data types.
- MySQL Connector API is implemented using pure Python and does not require any third-party library.
- You can download MySQL connector API from https://dev.mysql.com

Installation

- Type the following *pip* command from Python terminal.

```
pip  install mysql-connector-python
```

Using MySql database

- After installation, before writing database code, you can test the MySQL connector from Python shell using the following command:

```
>>> import mysql.connector
>>>
```

- If you do not get any error, it means that in your system, MySQL connector is installed successfully.
- Now, you can work with MySQL database using Python program.

Python Code to create MySQL database

```
import mysql.connector

db = mysql.connector.connect (
  host="localhost",
  user="root",
  password="bbp123"
)
cur = db.cursor ()

cur.execute ( "CREATE DATABASE bbp")

print ( "bbp database is created successfully")
```

Output

```
bbp database is created successfully
>>>
```

Python Code to create a table inside MySQL database

```
import mysql.connector

db = mysql.connector.connect (
  host="localhost",
  user="root",
  password="bbp123",
  database="bbp"
)

cur = db.cursor ()

cur.execute ( ''' CREATE TABLE stud  ( sno integer,
                                snm varchar ( 50)  ,
                                sper integer )
                                ''' )
print ( "Stud table is created")
```

Output

```
Stud table is created
>>>
```

Python Code to insert records in a table

```
import mysql.connector

db = mysql.connector.connect (
  host="localhost",
  user="root",
  password="bbp123",
  database="bbp"
)

cur = db.cursor ()
sql = "INSERT INTO stud  ( sno,snm,sper)  VALUES  ( 101,'XYZ',85.00)  "
cur.execute ( sql)
cur.execute ( "INSERT INTO stud  ( sno,snm,sper)  VALUES  ( 102,'ABC',45.00)  " )
db.commit ()

print ( "Two records are inserted")
```

Output

```
Two records are inserted
>>>
```

Python Code to retrieve records from a table

```
import mysql.connector

db = mysql.connector.connect (
  host="localhost",
  user="root",
  password="bbp123",
  database="bbp"
)

cur = db.cursor ()

cur.execute ( "SELECT * FROM stud")

result = cur.fetchall ()

for x in result:
  print (x)
```

Output

```
(101, 'XYZ', 85)
(102, 'ABC', 45)
>>>
```