Unit 5 GUI Programming and MySQL

Introduction

- The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit.
- Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems.
- Tkinter supports a range of Tcl/Tk versions, built either with or without thread support. The official Python binary release bundles Tcl/Tk 8.6 threaded.
- Tcl/Tk is not a single library but rather consists of a few distinct modules, each with separate functionality and its own official documentation.
- Python’s binary releases also ship an add-on module together with it.

Python GUI Programming - Tkinter

- It is the standard GUI toolkit for Python.
- It was written by Fredrik Lundh.
- Python when combined with Tkinter provides a fast and easy way to create GUI applications.
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Creating a GUI application using Tkinter is an easy task.
- Tkinter is an acronym for "Tk interface."
- Tkinter comes pre-installed with Python3.

Tkinter Modules

- Most applications will need the main tkinter module, as well as the tkinter.ttk module, which provides the modern themed widget set and API.
    - `from tkinter import *`
    - `from tkinter import ttk`

- Python offers multiple options for developing GUI (Graphical User Interface).
- 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 Tkinter is the fastest and easiest way to create GUI applications.

Tk()

- 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.

---

mainloop()

- It is used when the 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 until the window is closed.
- Example: `m.mainloop()`

---

pack() method

- 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.

pack() method

- It organizes the widgets relatively in blocks before placing them in the parent widget.
- Example:
  - `widget.pack(option=value, ...)`
    - **side**: Specifies which side of the parent widget to pack the widget against. Possible values are TOP, BOTTOM, LEFT, RIGHT.
    - **fill**: Determines if the widget should expand to fill any extra space. Can take values:
      - NONE (default): The widget keeps its natural size.

        - X: The widget expands horizontally to fill the space.
        - Y: The widget expands vertically to fill the space.
        - BOTH: The widget expands in both directions.
    - **expand**: If True, the widget expands to fill any extra space in the geometry manager's parent widget.
    - **padx, pady**: Specifies padding along the X and Y axes, respectively, adding space around the widget.
    - **anchor**: Specifies where the widget should be packed relative to the available space (e.g., N, S, E, W, CENTER).

---

place() method

- The place() method is another geometry manager used to position widgets within a parent widget. Unlike pack() and grid(), the place() method allows you to specify the exact position of a widget using absolute or relative coordinates.
- Example:
    - `widget.place(option=value, ...)`
        - **x**: The x-coordinate (horizontal position) of the widget, in pixels.
        - **y**: The y-coordinate (vertical position) of the widget, in pixels.
        - **relx**: Specifies the relative x-coordinate as a float (0.0 to 1.0) with respect to the width of the parent widget.
        - **rely**: Specifies the relative y-coordinate as a float (0.0 to 1.0) with respect to the height of the parent widget.
        - **anchor**: Specifies which part of the widget is placed at the x, y coordinate. Options are: N, S, E, W, NE, NW, SE, SW, CENTER.
        - **width**: Specifies the width of the widget in pixels.
        - **height**: Specifies the height of the widget in pixels.
        - **relwidth**: Specifies the relative width of the widget as a float (0.0 to 1.0).
        - **relheight**: Specifies the relative height of the widget as a float (0.0 to 1.0).

---

grid() method

- The grid() method is a geometry manager that organizes widgets in a table-like structure using a grid of rows and columns. Unlike pack() and place(), the grid() method allows widgets to be arranged in a more structured way, assigning them specific row and column locations.
- Example:
  - `widget.grid(option=value, ...)`
    - **row**: Specifies the row in which the widget should be placed (default is 0).
    - **column**: Specifies the column in which the widget should be placed (default is 0).
    - **rowspan**: Specifies how many rows the widget should span (default is 1).
    - **columnspan**: Specifies how many columns the widget should span (default is 1).
    - **sticky**: Determines where the widget should "stick" within the cell. It can take values like N, S, E, W, or combinations like NE, SW to specify alignment (e.g., top-left, bottom-right).
    - **padx**: Specifies horizontal padding (space) inside the cell.
    - **pady**: Specifies vertical padding (space) inside the cell.
    - **ipadx**: Specifies the internal horizontal padding (within the widget).
    - **ipady**: Specifies the internal vertical padding (within the widget).

Grid Manager

- The Grid geometry manager places the widgets in a 2-dimensional table, which consists of a number of rows and columns.
- The position of a widget is defined by a row and a column number.
- Widgets with the same column number and different row numbers will be above or below each other.
- Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside each other, i.e. to the left or the right.

- Using the grid manager means that you create a widget, and use the grid method to tell the manager in which row and column to place them.
- The size of the grid doesn't have to be defined, because the manager automatically determines the best dimensions for the widgets used.

Tkinter Variable Classes

- Variables can be used with most entry widgets to track changes to the entered value.
- The Checkbutton and Radiobutton widgets require variables to work properly.
- Some widgets (like text entry widgets, radio buttons, and so on) can be connected directly to application variables by using special options:
  - `variable`, `textvariable`, `onvalue`, `offvalue`, and `value`.
- Create a Tkinter variable:
  - `x = StringVar()` # Holds a string; default value ""
  - `x = IntVar()` # Holds an integer; default value 0
  - `x = DoubleVar()` # Holds a float; default value 0.0
  - `x = BooleanVar()` # Holds a boolean, returns 0 for False and 1 for True

---

Tkinter Widgets

- Tkinter provides various controls, such as buttons, labels, and text boxes used in a GUI application.
- These controls are commonly called widgets.
- Common widgets of Tkinter are:
  - **Label**
  - **Button**
  - **Canvas**
  - **combo-box**
  - **frame**
  - **level**
  - **check-button**

- **entry**
- **level-frame**
- **menu**
- **list-box**
- **menu button**
- **Message**
- **tk_option Menu**
- **progress-bar**
- **radio button**
- **scroll bar**
- **Separator**
- **tree-view** and many more.

This format is now ready for Notion with the original text intact.

Tkinter Widgets

1. **Label**
   - **Explanation**: A Label widget is used to display text or images in the GUI.
   - **Example**:

```
from tkinter import *
root = Tk()
label = Label(root, text="Hello, Tkinter!")
label.pack()
root.mainloop()
```

2. **Button**
   - **Explanation**: A Button widget is used to perform an action when clicked by the user.
   - **Example**:

```
from tkinter import *
def on_click():
    print("Button Clicked!")
root = Tk()
button = Button(root, text="Click Me", command=on_cli
ck)
button.pack()
root.mainloop()
```

3. **Canvas**
    - **Explanation**: A Canvas widget is used to draw shapes like lines, circles, and rectangles in the GUI.
    - **Example**:

```
from tkinter import *
root = Tk()
canvas = Canvas(root, width=200, height=100)
canvas.pack()
canvas.create_line(0, 0, 200, 100)
canvas.create_rectangle(50, 25, 150, 75, fill="blue")
root.mainloop()
```

4. **Combo-box**
    - **Explanation**: A Combo-box widget allows the user to select an option from a drop-down list.
    - **Example**:

```
from tkinter import *
from tkinter.ttk import Combobox
root = Tk()
combo = Combobox(root, values=["Option 1", "Option
2", "Option 3"])
combo.pack()
root.mainloop()
```

5. **Frame**

- **Explanation**: A Frame widget is a container that can hold other widgets, helping in organizing the layout.
- **Example**:

```
from tkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
label = Label(frame, text="Inside a Frame")
label.pack()
root.mainloop()
```

6. **Check-button**
   - **Explanation**: A Check-button widget allows the user to select one or more options by checking boxes.
   - **Example**:

```
from tkinter import *
root = Tk()
var = IntVar()
check = Checkbutton(root, text="Check Me", variable=v
ar)
check.pack()
root.mainloop()
```

7. **Entry**
   - **Explanation**: An Entry widget is used to accept a single line of text input from the user.
   - **Example**:

```
from tkinter import *
root = Tk()
entry = Entry(root)
entry.pack()
root.mainloop()
```

8. **List-box**
    - **Explanation**: A List-box widget is used to display a list of items from which the user can select.
    - **Example**:

```
from tkinter import *
root = Tk()
listbox = Listbox(root)
listbox.insert(1, "Item 1")
listbox.insert(2, "Item 2")
listbox.pack()
root.mainloop()
```

9. **Menu**
    - **Explanation**: A Menu widget is used to create drop-down menus in the application.
    - **Example**:

```
from tkinter import *
root = Tk()
menu = Menu(root)
root.config(menu=menu)
submenu = Menu(menu)
menu.add_cascade(label="File", menu=submenu)
submenu.add_command(label="New")
submenu.add_command(label="Exit", command=root.quit)
root.mainloop()
```

10. **Progress-bar**
    - **Explanation**: A Progress-bar widget is used to show the progress of a long-running operation.
    - **Example**:

```
from tkinter import *
from tkinter.ttk import Progressbar
root = Tk()
```

```
progress = Progressbar(root, orient=HORIZONTAL, lengt
h=200, mode='determinate')
progress.pack()
root.mainloop()
```

11. **Radio-button**

- **Explanation**: A Radio-button widget allows the user to select only one option from a set.
- **Example**:

```
from tkinter import *
root = Tk()
var = IntVar()
radio1 = Radiobutton(root, text="Option 1", variable=
var, value=1)
radio2 = Radiobutton(root, text="Option 2", variable=
var, value=2)
radio1.pack()
radio2.pack()
root.mainloop()
```

12. **Scroll-bar**

- **Explanation**: A Scroll-bar widget adds scrolling functionality to other widgets like text boxes or list boxes.
- **Example**:

```
from tkinter import *
root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)
listbox = Listbox(root, yscrollcommand=scrollbar.set)
for i in range(100):
    listbox.insert(END, "Item " + str(i))
listbox.pack(side=LEFT, fill=BOTH)
scrollbar.config(command=listbox.yview)
root.mainloop()
```

Python MySQL Guide

1. MySQL Driver

- A MySQL driver is required for Python to connect and communicate with a MySQL database.
- **Popular Driver**: `mysql-connector-python`

Installation

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

Importing the MySQL Connector

```
import mysql.connector
```

---

2. Creating a Connection to MySQL

Connection Parameters

- **host**: The server name (e.g., `localhost`).
- **user**: MySQL username (e.g., `root`).
- **password**: MySQL password.

Example: Creating a Connection

```
import mysql.connector

# Establish a connection
mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword"
)
```

```
if mydb.is_connected():
    print("Connection to MySQL was successful.")
```

3. Creating a Database

Steps to Create a Database

1. **Establish a connection.**
2. **Create a cursor object.**
3. **Execute the `CREATE DATABASE` SQL command.**

Example: Creating a Database

```
# Create a cursor object
mycursor = mydb.cursor()

# Create a new database
mycursor.execute("CREATE DATABASE IF NOT EXISTS mydatabas
e")
print("Database 'mydatabase' created.")
```

4. MySQL DML Statements (Data Manipulation Language)

Overview of DML Operations

- **DML Statements**: Allow for inserting, updating, selecting, and deleting data within tables.

Using DML Statements

1. **Connect to the specific database.**
2. **Create a cursor object.**
3. **Execute SQL statements for DML operations.**
4. **Use `commit()` to apply changes.**
5. **Use `fetchall()` or `fetchone()` to retrieve data.**

Example 1: Creating a Table

```
# Connect to the database
mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

# Create a cursor object
mycursor = mydb.cursor()

# Create a table named 'customers'
mycursor.execute("""
    CREATE TABLE IF NOT EXISTS customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
print("Table 'customers' created.")
```

Example 2: Inserting Data (DML: INSERT)

```
# Insert data into the 'customers' table
sql = "INSERT INTO customers (name, address) VALUES (%s, %
s)"
val = ("John Doe", "123 Apple St")
mycursor.execute(sql, val)

# Commit the transaction
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

Example 3: Selecting Data (DML: SELECT)

```
# Select all records from the 'customers' table
mycursor.execute("SELECT * FROM customers")

# Fetch all rows from the result
result = mycursor.fetchall()

# Display the results
print("Data in 'customers' table:")
for row in result:
    print(row)
```

Example 4: Updating Data (DML: UPDATE)

```
# Update a record in the 'customers' table
sql = "UPDATE customers SET address = %s WHERE name = %s"
val = ("456 Orange Blvd", "John Doe")
mycursor.execute(sql, val)

# Commit the transaction
mydb.commit()
print(mycursor.rowcount, "record(s) updated.")
```

Example 5: Deleting Data (DML: DELETE)

```
# Delete a record from the 'customers' table
sql = "DELETE FROM customers WHERE name = %s"
val = ("John Doe",)
mycursor.execute(sql, val)

# Commit the transaction
mydb.commit()
print(mycursor.rowcount, "record(s) deleted.")
```

5. Closing the Connection

After completing operations, always close the connection:

```
mydb.close()
print("Database connection closed.")
```

Summary

- **MySQL Driver**: Use `mysql-connector-python` for connection.
- **Create Connection**: Utilize `mysql.connector.connect()`.
- **Create Database**: Use `CREATE DATABASE` SQL command.
- **DML Statements**: Include INSERT, SELECT, UPDATE, and DELETE operations.
- **Close Connection**: Ensure to close the connection when done.