Tkinter

Key Concepts:

1. **Tkinter and Python Library:**

- Tkinter is a standard module in Python that provides classes and methods for creating GUIs.
- It interfaces with Tcl/Tk, an open-source toolkit, hence the name 'Tkinter' (Tk Interface).
- Tkinter is pronounced as "tee-kay-inter".

2. **Core Components of Tkinter:**
   - **Tkinter Library:** Part of Python's standard library, eliminating the need for separate installations.
   - **Tcl (Tool Command Language):** The scripting language underlying Tkinter. Python code using Tkinter is eventually translated into Tcl commands.
   - **Tk:** A GUI toolkit used by Tcl. The combination is often referred to as Tk/Tcl and pronounced as "tick" or "tee-kay".
3. **Cross-Language Usability:**
   - While we focus on Python's use of Tkinter, it's noteworthy that languages like Ruby also utilize Tk.

Advantages of Using Tkinter:

1. **Ease of Learning:**
   - Tkinter is user-friendly, allowing the creation of GUIs with minimal code.
   - It has a more accessible learning curve compared to other GUI frameworks.
2. **Cross-Platform Compatibility:**
   - GUIs built with Tkinter can run seamlessly across various operating systems like Windows, macOS, and Linux.
3. **Bundled with Python:**
   - Tkinter comes with the standard Python distribution, so no additional installation is required.
4. **Applicability:**
   - Suitable for beginners and for those who want to quickly develop GUI applications in Python.

For More,

- https://www.tkdocs.com/
- https://tcl.tk/doc/

Master Window:

The master window in Tkinter is the main window of a Tkinter application. It serves as the primary container in which other widgets like buttons, labels, entry fields, etc., are placed and displayed. Understanding how to create and manage the master window is fundamental to using Tkinter effectively.

Creating a Master Window

When you start a Tkinter application, the first step is to create the master window. This is done using the `Tk()` class. Here's a basic example:

```
1 import tkinter as tk
2
```

```
# Create the master window
master = tk.Tk()

# Set the title of the window
master.title("My Tkinter Application")

# Define the size of the window (optional)
master.geometry("400x300")

# Start the event loop
master.mainloop()

```

In this example, `master` is the master window. We set its title and size, and then start Tkinter's event loop with `master.mainloop()`. The event loop is crucial as it waits for events (like button clicks) and processes them as long as the window is open.

Widgets

Widgets are the building blocks of a graphical user interface (GUI) in Tkinter, Python's standard GUI toolkit. Each widget in Tkinter is used to create various GUI elements like buttons, labels, text boxes, and more. Understanding these widgets and how to use them is essential for designing and developing effective GUI applications. Here's an overview of some common widgets in Tkinter:

1. Labels:

The Label widget in Tkinter is one of the simplest and most commonly used widgets. It's primarily used for displaying text or images. The label can be customized in various ways to suit the needs of your GUI application.

Example,

```
from tkinter import *

# Create the main window
root = Tk()
root.title("Label Example")

# Create a label widget
label = Label(root, text="Hello, Tkinter!")

# Display the label
label.pack()

# Run the application
root.mainloop()

```

Customizing the Label

The Label widget can be customized with various options:

- **Text:** The `text` property sets the text to be displayed.
- **Fonts and Colors:** You can change the font (`font`), the foreground color (`fg`), and the background color (`bg`).
- **Images:** Instead of text, you can display an image using the `image` property.
- **Justification and Alignment:** Text inside the label can be aligned using `anchor`, `justify`, and `padx`, `pady` for padding.
- **Relief Style:** The border of the label can have different styles like `flat`, `raised`, `sunken`, etc., using the `relief` property.
- **Border Width:** Use `bd` or `borderwidth` to set the width of the border.

```
from tkinter import *

# Create the main window
root = Tk()
root.title("Custom Label Example")

# Create a customized label
custom_label = Label(root,
                     text="Broadridge",
                     fg="white",
                     bg="blue",
                     font=("Helvetica", 16),
                     padx=10,
                     pady=10,
                     borderwidth=2,
                     relief="solid")

# Display the label
custom_label.pack()

# Run the application
root.mainloop()

```

2. Button

The Button widget in Python, specifically in the Tkinter GUI toolkit, is a fundamental and widely-used widget. It allows users to perform an action when clicked. Understanding how to use and customize buttons is crucial for interactive and functional GUI applications.

Here's how you can create a basic button in a Tkinter application:

```
from tkinter import *


def on_button_click():

```

```
    print("Button was clicked!")


# Create the main window
root = Tk()
root.title("Button Example")

# Create a button widget
button = Button(root, text="Click Me", command=on_button_click)

# Display the button
button.pack()

# Run the application
root.mainloop()

```

In this example, `on_button_click` is a function that gets executed when the button is clicked.

Customizing the Button

The Button widget can be customized in various ways:

- **Text:** Change the text on the button with the `text` attribute.
- **Command:** The `command` attribute specifies the function to be called when the button is clicked.
- **Colors and Fonts:** Customize with `fg` (foreground color), `bg` (background color), and `font`.
- **Size:** Adjust the size using `height` and `width`.
- **Images:** Use `image` to display an image on the button.
- **State:** Control the button state (`normal`, `active`, `disabled`) with the `state` attribute.
- **Styling:** Add a visual style using `relief` (e.g., `raised`, `sunken`) and `borderwidth`.

```
import tkinter as tk


def on_custom_button_click():
    print("Custom button clicked!")


# Create the main window
root = tk.Tk()
root.title("Custom Button Example")

# Create a customized button
custom_button = tk.Button(
    root,
    text="Custom Button",
    command=on_custom_button_click,
    fg="red",
    bg="white",
```

```
    font=("Arial", 12),
    padx=10,
    pady=10,
    borderwidth=2,
    relief="groove",
)

# Display the button
custom_button.pack()

# Run the application
root.mainloop()

```

3. Entry Widget

The Entry widget in Tkinter is a fundamental widget used in GUI applications to accept single-line text input from the user. It's straightforward to use and can be customized to suit the needs of your application.

Example of Entry Widget,

```
import tkinter as tk

# Function to get Entry input
def retrieve_input():
    input_value = entry.get()
    print(input_value)


# Create the main window
root = tk.Tk()
root.title("Entry Widget Example")

# Create an Entry widget
entry = tk.Entry(root)

# Display the Entry widget
entry.pack()

# Create a Button to retrieve input
submit_button = tk.Button(root, text="Submit", command=retrieve_input)
submit_button.pack()

# Run the application
root.mainloop()

```

In this example, the `Entry` widget is used for text input, and a `Button` widget is used to retrieve and print the input.

Customizing the Entry Widget

The Entry widget can be customized with various options:

- **Width:** Control the width of the widget with the `width` attribute.
- **Font and Colors:** Customize with `font`, `fg` (foreground color), and `bg` (background color).
- **Border:** Set the border width and relief.
- **Show:** Use `show="*"` for password fields to mask the input.
- **Text Variable:** Bind the widget to a `StringVar()` for dynamic updates.

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Custom Entry Widget")

# Widgets are Rely on "Linked Variables"
# Examples:
# ivar = IntVar() -- Takes int input
# svar = StringVar() -- Takes string input
# dvar = DoubleVar() -- Takes double input
# bvar = BooleanVar() -- Takes True/False input

# Create a StringVar to store text
text_var = tk.StringVar()

# Create a customized Entry widget
entry = tk.Entry(
    root,
    font=("Arial", 14),
    fg="blue",
    bg="lightgray",
    width=20,
    borderwidth=2,
    textvariable=text_var,
)

# Display the Entry widget
entry.pack()


# Function to get Entry input
def retrieve_input():
    print(text_var.get())


# Create a Button to retrieve input
submit_button = tk.Button(root, text="Submit", command=retrieve_input)
submit_button.pack()

# Run the application
root.mainloop()
```

```
43
```

Example, Addition of two no.s demostrating Linked Variables.

```python
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Addition of Two Nos")

# Widgets are Rely on "Linked Variables"
# Examples:
# ivar = IntVar() -- Takes int input
# svar = StringVar() -- Takes string input
# dvar = DoubleVar() -- Takes double input
# bvar = BooleanVar() -- Takes True/False input

# Create a StringVar to store text
ivar1 = tk.IntVar()
ivar2 = tk.IntVar()

# Create a customized Entry widget
entry1 = tk.Entry(
    root,
    font=("Arial", 14),
    fg="blue",
    bg="lightgray",
    width=20,
    borderwidth=2,
    textvariable=ivar1,
)

entry2 = tk.Entry(
    root,
    font=("Arial", 14),
    fg="blue",
    bg="lightgray",
    width=20,
    borderwidth=2,
    textvariable=ivar2,
)

# Display the Entry widget
entry1.pack()
entry2.pack()


# Function to get Entry input
def retrieve_input():
    print(ivar1.get() + ivar2.get())

```

```
48
49  # Create a Button to retrieve input
50  submit_button = tk.Button(root, text="Submit", command=retrieve_input)
51  submit_button.pack()
52
53  # Run the application
54  root.mainloop()
55
```

4. Text

The Text widget in Tkinter is a versatile widget used in GUI applications for multi-line text input or display. Unlike the Entry widget, which is for single-line text, the Text widget can handle a larger amount of text, making it suitable for applications like text editors, code editors, or chat applications.

Basic Usage of Text Widget

Here's how to create and use a Text widget in a Tkinter application:

```
1   import tkinter as tk
2
3   def retrieve_text():
4       input_text = text_widget.get("1.0", tk.END)
5       print(input_text)
6
7   # Create the main window
8   root = tk.Tk()
9   root.title("Text Widget Example")
10
11  # Create a Text widget
12  text_widget = tk.Text(root, height=10, width=40)
13
14  # Display the Text widget
15  text_widget.pack()
16
17  # Create a Button to retrieve text
18  submit_button = tk.Button(root, text="Submit", command=retrieve_text)
19  submit_button.pack()
20
21  # Run the application
22  root.mainloop()
```

In this example, `text_widget` is a multi-line text area, and the `retrieve_text` function prints the content of the text widget when the button is clicked.

Understanding Text Widget Indexing

Text in the Text widget is indexed with line and column numbers. The first character in the text widget is at `"1.0"` (line 1, column 0). The `tk.END` constant refers to the position just after the last character in the text widget.

Customizing the Text Widget

The Text widget can be customized in various ways:

- **Width and Height:** `width` and `height` attributes define the size.
- **Scrolling:** Can be combined with a `Scrollbar` widget for vertical or horizontal scrolling.
- **Font and Colors:** Customize with `font`, `fg` (foreground color), and `bg` (background color).
- **Border:** Adjust border width and relief.
- **Wrap:** Control line wrapping with the `wrap` attribute (`tk.WORD`, `tk.CHAR`, or `tk.NONE`).

Example with Customizations and Scrollbar

Here's a more customized example with a scrollbar:

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Text Widget with Scrollbar")

# Create a Text widget
text_widget = tk.Text(root, height=10, width=40, wrap=tk.WORD)

# Create a Scrollbar and set it to text_widget
scrollbar = tk.Scrollbar(root, command=text_widget.yview)
text_widget.configure(yscrollcommand=scrollbar.set)

# Pack Scrollbar and Text widget
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
text_widget.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)


# Function to retrieve text
def retrieve_text():
    print(text_widget.get("1.0", tk.END))


# Create a Button to retrieve text
submit_button = tk.Button(root, text="Submit", command=retrieve_text)
submit_button.pack()

# Run the application
root.mainloop()

```

5. Frame

The Frame widget in Tkinter is a container widget used to organize and group other widgets within a Tkinter application. Frames are particularly useful for complex GUIs, as they help in managing the layout by dividing the interface into sections.

Basic Usage of Frame Widget

Here's how to create and use a Frame widget in a Tkinter application:

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Frame Example")

# Create a Frame widget
frame = tk.Frame(root, bg="lightgray", bd=2, relief=tk.RAISED)

# Pack the Frame into the root window
frame.pack(padx=10, pady=10)

# Adding widgets to the frame
label = tk.Label(frame, text="I'm inside a Frame!")
label.pack(padx=5, pady=5)

button = tk.Button(frame, text="Click Me")
button.pack(padx=5, pady=5)

# Run the application
root.mainloop()
```

In this example, a `Frame` is created with a light gray background, a border width of 2, and a raised relief. A `Label` and a `Button` are added inside the frame.

Customizing the Frame Widget

Frames can be customized in various ways:

- **Background Color:** The `bg` attribute sets the background color.
- **Border Width and Relief:** `bd` sets the border width, and `relief` sets the relief style (e.g., `tk.FLAT`, `tk.RAISED`, `tk.SUNKEN`).
- **Size:** The `width` and `height` attributes define the size.
- **Padding:** `padx` and `pady` inside the `pack` method can be used for internal padding.

Organizing Widgets with Frames

Frames are essential for organizing widgets. By placing widgets in frames and then arranging these frames, you can achieve complex GUI layouts. Here's an example of using multiple frames:

```
# Create the main window
root = tk.Tk()
```

```
root.title("Multiple Frames Example")

# Top Frame
top_frame = tk.Frame(root, bg="blue", bd=5, relief=tk.RAISED)
top_frame.pack(fill=tk.X)

label_top = tk.Label(top_frame, text="Top Frame", fg="white", bg="blue")
label_top.pack()

# Bottom Frame
bottom_frame = tk.Frame(root, bg="green", bd=5, relief=tk.SUNKEN)
bottom_frame.pack(fill=tk.X)

label_bottom = tk.Label(bottom_frame, text="Bottom Frame", fg="white", bg="green")
label_bottom.pack()

button_bottom = tk.Button(bottom_frame, text="Click Me")
button_bottom.pack(pady=5)

# Run the application
root.mainloop()
```

6. Check Box

The Checkbox widget in Tkinter, known as `Checkbutton`, is a widely-used widget that allows users to make selections. Each Checkbutton can be either checked or unchecked, making them ideal for presenting options where the user can choose multiple items.

Basic Usage of Checkbutton

Here's how to create a simple Checkbutton in Tkinter:

```
import tkinter as tk

def display_selection():
    selection = "Selected" if var.get() else "Not Selected"
    print(selection)

# Create the main window
root = tk.Tk()
root.title("Checkbox Example")

# Create a variable to hold the state of the checkbox
var = tk.IntVar()

# Create a Checkbutton
checkbox = tk.Checkbutton(root, text="Check me", variable=var,
command=display_selection)

# Display the Checkbutton
checkbox.pack()
```

```
19
20 # Run the application
21 root.mainloop()
```

In this example, `var` is an `IntVar`, a special Tkinter variable to hold the state of the checkbox (0 for unchecked, 1 for checked). The `display_selection` function prints the current state of the checkbox.

Customizing the Checkbutton

You can customize Checkbuttons in several ways:

- **Text:** Set the label of the Checkbutton using the `text` attribute.
- **Variable:** Use a Tkinter variable (`IntVar`, `StringVar`, etc.) to track the state.
- **Command:** Assign a function to be called when the state changes.
- **Colors and Fonts:** Customize the text and background colors with `fg` and `bg`, and the font with the `font` attribute.

Example with Multiple Checkbuttons

Here's an example showing how to create and use multiple Checkbuttons:

```
1  import tkinter as tk
2
3
4  def show_selection():
5      count = var1.get() + var2.get()  # Each variable is 1 if checked, 0 if not
6      message = f"{count} checkbox(es) selected"
7      print(message)
8
9
10 # Create the main window
11 root = tk.Tk()
12 root.title("Multiple Checkboxes Example")
13
14 # Variables to store the states of the checkboxes
15 var1 = tk.IntVar()
16 var2 = tk.IntVar()
17
18 # Create two Checkbuttons
19 checkbox1 = tk.Checkbutton(root, text="Checkbox 1", variable=var1)
20 checkbox2 = tk.Checkbutton(root, text="Checkbox 2", variable=var2)
21
22 # Display the Checkbuttons
23 checkbox1.pack()
24 checkbox2.pack()
25
26 # Create a Button that prints how many checkboxes are selected
27 button = tk.Button(root, text="Show Selection", command=show_selection)
28
29 # Display the Button
30 button.pack()
```

```
31
32 # Run the application
33 root.mainloop()
34
```

7. Radio Button

Radio buttons in Tkinter, implemented as `Radiobutton` widgets, allow users to select one option from a group of choices. Unlike checkboxes, radio buttons are mutually exclusive - when one is selected, any previously selected button in the group is deselected.

Basic Usage of Radiobutton

Here's an example to create and use radio buttons in Tkinter:

```
1  import tkinter as tk
2
3  def show_choice():
4      print(f"Selected Option: {var.get()}")
5
6  # Create the main window
7  root = tk.Tk()
8  root.title("Radio Button Example")
9
10 # Variable to store the currently selected option
11 var = tk.IntVar()
12
13 # Create radio buttons
14 radio1 = tk.Radiobutton(root, text="Option 1", variable=var, value=1,
   command=show_choice)
15 radio2 = tk.Radiobutton(root, text="Option 2", variable=var, value=2,
   command=show_choice)
16 radio3 = tk.Radiobutton(root, text="Option 3", variable=var, value=3,
   command=show_choice)
17
18 # Display the radio buttons
19 radio1.pack()
20 radio2.pack()
21 radio3.pack()
22
23 # Run the application
24 root.mainloop()
```

In this example:

- `var` is an `IntVar`, a special Tkinter variable to hold the value of the selected radio button.
- Three `Radiobutton` widgets (`radio1`, `radio2`, `radio3`) are created, each with a different `value`.
- The `command` option in each `Radiobutton` is set to `show_choice`, which prints the currently selected option when any radio button is clicked.
- All radio buttons are linked to the same variable (`var`), ensuring mutual exclusivity.

Customizing Radiobuttons

- **Text:** Use the `text` attribute to set the label of the radio button.
- **Value:** The `value` attribute assigns a unique value to each radio button in the group.
- **Colors, Fonts, and More:** Like other Tkinter widgets, `Radiobutton` can be customized with `fg`, `bg`, `font`, etc.

8. List Box

The Listbox widget in Tkinter is used to display a list of items from which the user can select one or more items. It's especially useful for presenting a list of options or data in a confined space within your GUI application.

Basic Usage of Listbox

Here's a simple example to demonstrate how to create and use a Listbox in Tkinter:

```
import tkinter as tk

def show_selection():
    selected_indices = listbox.curselection()
    selected_items = [listbox.get(i) for i in selected_indices]
    print("Selected items:", selected_items)

# Create the main window
root = tk.Tk()
root.title("Listbox Example")

# Create a Listbox widget
listbox = tk.Listbox(root)

# Add items to the Listbox
for item in ["Item 1", "Item 2", "Item 3", "Item 4"]:
    listbox.insert(tk.END, item)

# Display the Listbox
listbox.pack()

# Create a Button to show selected item(s)
button = tk.Button(root, text="Show Selection", command=show_selection)
button.pack()

# Run the application
root.mainloop()
```

In this example:

- A `Listbox` widget (`listbox`) is created and populated with items using the `insert` method.

- The `curselection` method is used to get the indices of selected items.
- The `get` method retrieves the selected items, which are printed when the button is clicked.

Customizing the Listbox

You can customize the Listbox in several ways:

- **Multiple Selections:** Set `selectmode` to `tk.MULTIPLE` to allow multiple selections.
- **Scrolling:** Combine with a `Scrollbar` widget for long lists.
- **Size:** Use `height` and `width` to control the size.
- **Font and Colors:** Customize with `font`, `fg`, and `bg`.

Example with Scrollbar and Multiple Selections

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Listbox with Scrollbar")

# Create a Listbox with multiple selection mode
listbox = tk.Listbox(root, selectmode=tk.MULTIPLE)

# Add a Scrollbar
scrollbar = tk.Scrollbar(root)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Attach Listbox to Scrollbar
listbox.config(yscrollcommand=scrollbar.set)
scrollbar.config(command=listbox.yview)

# Add items to the Listbox
for item in range(50):  # Adding 50 items
    listbox.insert(tk.END, f"Item {item+1}")


def show_selection():
    selected_indices = listbox.curselection()
    selected_items = [listbox.get(i) for i in selected_indices]
    print("Selected items:", selected_items)


# Button and function to show selected items
button = tk.Button(root, text="Show Selection", command=show_selection)
button.pack()

# Run the application
root.mainloop()

```

9. Scroll Bar

The Scrollbar widget in Tkinter is used to add scrolling capability to other widgets, like `Listbox`, `Text`, or `Canvas`. It's especially useful when dealing with content that exceeds the visible area of these widgets.

Basic Usage of Scrollbar with a Listbox

Here's an example of how to add a vertical scrollbar to a Listbox:

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Scrollbar Example")

# Create a Scrollbar
scrollbar = tk.Scrollbar(root)

# Create a Listbox and attach it to the Scrollbar
listbox = tk.Listbox(root, yscrollcommand=scrollbar.set)
for i in range(100):
    listbox.insert(tk.END, f"Item {i+1}")

# Configure the Scrollbar
scrollbar.config(command=listbox.yview)

# Pack the widgets
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Run the application
root.mainloop()

```

In this example:

- A `Scrollbar` widget (`scrollbar`) is created.
- A `Listbox` widget (`listbox`) is created and its `yscrollcommand` is linked to the `Scrollbar`.
- The `Scrollbar` is configured with the `command` attribute to control the `yview` (vertical view) of the `Listbox`.
- Both widgets are packed with the `Scrollbar` on the right side and the `Listbox` on the left.

Customizing the Scrollbar

- **Orientation:** By default, a scrollbar is vertical. Set `orient=tk.HORIZONTAL` for a horizontal scrollbar.
- **Size and Colors:** Customize with `width`, `bg`, and `troughcolor`.

Scrollbar with a Text Widget

Here's an example with a Text widget:

```python
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Scrollbar with Text Widget")

# Create a Scrollbar
scrollbar = tk.Scrollbar(root)

# Create a Text widget and attach it to the Scrollbar
text = tk.Text(root, yscrollcommand=scrollbar.set)
scrollbar.config(command=text.yview)

# Pack the widgets
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Run the application
root.mainloop()

```

10. Menu Bar

Creating a menubar in Tkinter involves using the `Menu` widget. A menubar is typically placed at the top of an application window and contains various menus, each of which can contain menu items, such as commands, checkboxes, radio buttons, and submenus.

Basic Usage of Menubar

Here's a simple example of creating a menubar with a few menus and menu items:

```python
import tkinter as tk


def menu_command():
    print("Menu item clicked")


# Create the main window
root = tk.Tk()
root.title("Menubar Example")

# Create a menubar
menubar = tk.Menu(root)

# Create a menu and add it to the menubar
file_menu = tk.Menu(menubar, tearoff=0)
file_menu.add_command(label="New", command=menu_command)
file_menu.add_command(label="Open", command=menu_command)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)
```

```
menubar.add_cascade(label="File", menu=file_menu)

# Create another menu
edit_menu = tk.Menu(menubar, tearoff=0)
edit_menu.add_command(label="Cut", command=menu_command)
edit_menu.add_command(label="Copy", command=menu_command)
edit_menu.add_command(label="Paste", command=menu_command)
menubar.add_cascade(label="Edit", menu=edit_menu)

# Attach the menubar to the window
root.config(menu=menubar)

# Run the application
root.mainloop()

```

In this example:

- A `Menu` widget (`menubar`) is created and attached to the root window.
- Two menus (`file_menu` and `edit_menu`) are created and added to the menubar with `add_cascade`. Each menu item is associated with a `command`.
- The `add_command` method adds individual commands to each menu. `add_separator` adds a separator line.
- The `tearoff` attribute is set to 0 to disable the ability to tear off the menu.

Customizing Menubar

- **Submenus:** Menus can have nested submenus by creating a `Menu` and adding it to a parent menu using `add_cascade`.
- **Checkbuttons and Radiobuttons:** Use `add_checkbutton` and `add_radiobutton` to add these types of menu items.
- **Shortcut Keys:** Assign shortcut keys using the `accelerator` attribute in `add_command`.

```
import tkinter as tk


def menu_command():
    print("Menu item clicked")


# Create the main window
root = tk.Tk()
root.title("Menubar Example")

# Create a menubar
menubar = tk.Menu(root)

# Create a menu with a submenu
file_menu = tk.Menu(menubar, tearoff=0)
```

```
submenu = tk.Menu(file_menu, tearoff=0)
submenu.add_command(label="Subitem 1", command=menu_command)
submenu.add_command(label="Subitem 2", command=menu_command)
file_menu.add_cascade(label="Submenu", menu=submenu)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=file_menu)

root.config(menu=menubar)
root.mainloop()

```

Integrating Tkinter with Databases

Creating a simple Tkinter application with SQLite database integration to store contact information involves a few steps. You'll create a GUI for inputting contact details and use SQLite to store this data persistently. Here's a basic example:

Step 1: Set Up SQLite Database

First, you'll need to create a SQLite database and a table to store contacts. This can be done using SQLite's command line tools or programmatically in Python. Here's how to do it in Python:

```
import sqlite3

# Connect to SQLite Database
conn = sqlite3.connect("contacts.db")
cursor = conn.cursor()

# Create table
cursor.execute(
    """
CREATE TABLE IF NOT EXISTS contacts (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    phone TEXT NOT NULL
)
"""
)

conn.commit()
conn.close()

```

Run this script once to set up your database and table.

Step 2: Create Tkinter GUI

Now, create the Tkinter application:

```
import tkinter as tk
```

```python
import sqlite3


def add_contact():
    name = name_entry.get()
    phone = phone_entry.get()

    # Insert into database
    conn = sqlite3.connect("contacts.db")
    cursor = conn.cursor()
    cursor.execute("INSERT INTO contacts (name, phone) VALUES (?, ?)", (name, phone))
    conn.commit()
    conn.close()

    # Clear the entry fields
    name_entry.delete(0, tk.END)
    phone_entry.delete(0, tk.END)


def show_contacts():
    # Connect to database and fetch contacts
    conn = sqlite3.connect("contacts.db")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM contacts")
    records = cursor.fetchall()
    conn.close()

    # Create a new window to show contacts
    contact_window = tk.Toplevel()
    contact_window.title("Contacts")

    # Display each contact in the new window
    for idx, record in enumerate(records):
        tk.Label(contact_window, text=f"{record[1]} - {record[2]}").grid(
            row=idx, column=0
        )


# Set up the main window
root = tk.Tk()
root.title("Contact Book")

# Create a menu
menu = tk.Menu(root)
root.config(menu=menu)

# Add 'File' menu
file_menu = tk.Menu(menu, tearoff=0)
menu.add_cascade(label="File", menu=file_menu)
file_menu.add_command(label="Show Contacts", command=show_contacts)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)
```

```
54
55  # Create and pack widgets for adding contacts
56  tk.Label(root, text="Name").pack()
57  name_entry = tk.Entry(root)
58  name_entry.pack()
59
60  tk.Label(root, text="Phone").pack()
61  phone_entry = tk.Entry(root)
62  phone_entry.pack()
63
64  add_button = tk.Button(root, text="Add Contact", command=add_contact)
65  add_button.pack()
66
67  # Run the application
68  root.mainloop()
69
```

How It Works

- This script creates a simple GUI with two entry fields for name and phone number, and a button to add the contact to the SQLite database.
- When you click the "Add Contact" button, the `add_contact` function is triggered. It retrieves the name and phone number from the entry fields and inserts this data into the `contacts` table in the SQLite database.
- After adding the contact, the entry fields are cleared.
- The `show_contacts` function fetches all contacts from the SQLite database and opens a new window (`Toplevel`) to display them.
- The `Menu` widget is used to create a menu bar at the top of the main window. It contains a "File" menu with an option to "Show Contacts" and to "Exit" the application.
- The "Show Contacts" menu item is linked to the `show_contacts` function, which displays the contacts.

Extending the Application

As before, you can further enhance this application by:

- Implementing edit and delete functionality for contacts.
- Adding scrollbars to the contacts window if the list is long.
- Improving the UI with better layout management.