Graphical User Interfaces

GUI Programming and Design

- Graphical user interfaces (GUIs) are a standard application for object-oriented programming.
- If the user can’t figure the interface out, it doesn’t matter how good the program is.
- An interface must be usable:
  - Include the information users need; leave out the information they don’t need.
  - Be consistent from one window to another.
  - Use commonly known interface patterns.
  - Give feedback.
  - Put the user in control.

Tkinter

- GUI library used by many programming languages for developing GUIs on Windows, Mac and UNIX.
- De facto standard for developing GUI programs in Python

Common Simple Widgets

- Button - simple button
- Entry - text entry field
- Label - to display text or image

- Many more available:
  - Canvas
  - Checkbutton
  - Frame
  - Menu
  - Menubutton
  - Message
  - Radiobutton
  - Text

Widget Demo

from tkinter import *

class WidgetDemo:
    def __init__(self, window):
        window.title("Widgets demo")
        cbBold = Checkbutton(window, text = "Bold")
        rbRed = Radiobutton(window, text = "Red", bg = "red")
        rbYellow = Radiobutton(window, text = "Yellow", bg = "yellow")

        cbBold.pack()
        rbRed.pack()
        rbYellow.pack()

if __name__ == "__main__":
    root = Tk()
    app = WidgetDemo(root)
    root.mainloop()
Geometry Managers

- Geometry managers are used to place widgets inside a container
- 3 options:
  - pack
  - grid
  - place

**Only use one manager per container!**
- Use frame as a subcontainer if needed

Pack Manager

- By default:
  - One widget per line
  - Widget is only as big as required to display contents

**Options:**
- Place side by side
- fill to fill container by X, Y or BOTH
- expand to assign additional space

Pack Manager - One per line

```python
from tkinter import *

class PackManagerDemo:
    def __init__(self, window):
        window.title("Pack Manager Demo")
        Label(window, text = "Blue", bg = "blue").pack()
        Label(window, text = "Red", bg = "red").pack(fill = BOTH, expand = 1)
        Label(window, text = "Green", bg = "green").pack()

if __name__ == '__main__':
    root = Tk()
    app = PackManagerDemo(root)
    root.mainloop()
```

Pack Manager - Side by Side

```python
from tkinter import *

class PackManagerDemo:
    def __init__(self, window):
        window.title("Pack Manager Demo")
        Label(window, text = "Blue", bg = "blue").pack(side=LEFT)
        Label(window, text = "Red", bg = "red").pack(fill=BOTH, expand=1, side=LEFT)
        Label(window, text = "Green", bg = "green").pack(fill=BOTH, side=LEFT)

if __name__ == '__main__':
    root = Tk()
    app = PackManagerDemo(root)
    root.mainloop()
```

Grid Manager

- Places widgets into an invisible grid
- Can specify row and column
- Combine using rowspan and columnspan
- Can place within a cell using sticky
  - S, N, E, W, NW, NE, SW, SE
- Pad cell with optional space
  - padx, pady
- Pad inside widget borders
  - ipadx, ipady

Grid Manager Example

```python
from tkinter import *

class GridManagerDemo:
    def __init__(self, window):
        window.title("Grid Manager Demo")
        message = Message(window, text = "This message occupies three rows and two columns")
        Label(window, text = "First name: ", grid(row=1, column=1, rowspan=3, columnspan=2)
        Entry(window).grid(row=1, column=4, padx=5, pady=5)
        Label(window, text = "Last name: ", grid(row=2, column=3)
        Entry(window).grid(row=2, column=4)
        Button(window, text = "Get Name ", grid(row=3, padx=5, pady=5, column=4, sticky=E)

if __name__ == '__main__':
    root = Tk()
    app = GridManagerDemo(root)
    root.mainloop()
```
More structure

- Can create substructure using **Frame**
- A widget for holding widgets
  - Geometry manager
  - Put on a window

**Frame example**

```python
class FrameDemo:
    def __init__(self, window):
        window.title("Frame Demo")
        label = Label(window, text="Same text is written here")
        label.pack()
        frame = Frame(window)
        frame.pack()
        Button(frame, text = "ok").pack(side=LEFT)
        Button(frame, text = "not ok").pack(side=LEFT)

if __name__ == '__main__':
    root = Tk()
    app = FrameDemo(root)
    root.mainloop()
```

**Phone Dialer**

```python
from tkinter import *

class Dialer:
    def __init__(self, window):
        window.title("Phone dialer")
        Label(window, text="Number dialed...").grid(row=1)
        dialed = Entry(window, width=10)
        dialed.grid(row=1, column=1)
        buttons = Frame(window)
        for i in range(0, 9):
            Button(buttons, text=i+1).grid(row=i//3, column=i%3)
        Button(buttons, text=0).grid(row=3, column=0)
        buttons.grid(row=2, columnspan=2)

if __name__ == '__main__':
    root = Tk()
    app = Dialer(root)
    root.mainloop()
```

**Interactive Behavior**

- When a user clicks a button, we’d like something to happen
- How?
  - Bind a **function/method** to the pressing of the button using **command**
- For example:
  ```python
  ok_button = Button(window, text="OK", command=self.processOk)
  ok_button.pack()
  cancel_button = Button(window, text="Cancel", command=self.processCancel)
  cancel_button.pack()
  ```
- Interpreter will generate error if the function/method does not exist at runtime

```python
from tkinter import *

class Process_Button_Example:
    def __init__(self, window):
        window.title("Process Button Example")
        ok_button = Button(window, text="OK", command=self.processOk)
        ok_button.pack()
        cancel_button = Button(window, text="Cancel", command=self.processCancel)
        cancel_button.pack()

    def processOk(self):
        print('You clicked the OK button')

    def processCancel(self):
        print('You clicked the cancel button')

if __name__ == '__main__':
    root = Tk()
    app = Process_Button_Example(root)
    root.mainloop()
```
Behind the Scenes: Events

- Actions taken in the GUI create **events**
- Events are **handled** using **event handlers**
- Adding a **command** to a widget associates the specified event handler with events caused by that widget

Updating GUI Elements

- Processing events often requires us to update information on screen
- Generally information in the console is for the programmer, not the user
- Widgets may have associated variables
  - `StringVar`
  - `IntVar`
  - `DoubleVar`
- Each has `set()` and `get()` methods

Change Label Demo...

```python
from tkinter import *

class Dialer:
    def _init__(self, window):
        window.title("Phone dialer")
        Label(window, text="Number dialed...").grid(row=1)
        dialed = Entry(window, width=10)
        dialed.grid(row=1, column=1)
        buttons = Frame(window)
        for i in range(0, 9):
            Button(buttons, text=i+1).grid(row=i//3, column=i%3)
        Button(buttons, text=0).grid(row=3, column=1)
        buttons.grid(row=2, columnspan=2)

if __name__ == '__main__':
    root = Tk()
    app = Dialer(root)
    root.mainloop()
```