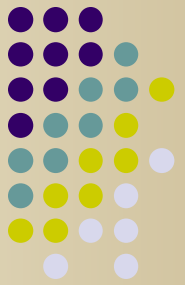
ENGR/CS 101 CS Session Lecture 15

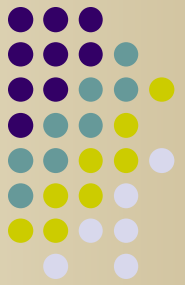


- Log into Windows/ACENET (reboot if in Linux)
- Use web browser to go to session webpage
  http://csserver.evansville.edu/~hwang/f14-courses/cs101.html
- Right-click on lecture15.py link. Save link/target to folder where your other CS 101 programs are.
- Browse to lecture15.py, right-click on it and select Edit with IDLE.

Outline

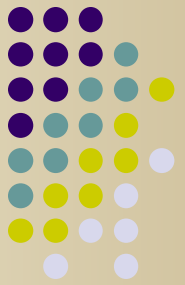
- Homework 3 out, due on Wednesday, December 3 (i.e. after Thanksgiving)
- Building GUIs in Python
  - Widgets
  - Layouts

Windowing Systems



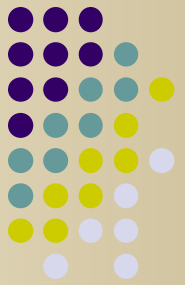
- WIMP interface: Window, Icons, Menus, Pointers
- Abstract actual hardware into concepts of screen, keyboard, and mouse
- Device drivers translate abstract commands into actual hardware commands
- USB has become a meta-abstraction for many peripherals

Window Management



- Several ways to organize windows
  - Each application could be responsible for its own windows. Not efficient.
  - Operating system manages windows. E.g. Windows, old MacOS
  - Separate management server, e.g. X Windows. Generic across operating systems, e.g. Unix, MacOS X.
- Generally, client-server architecture; often network-based

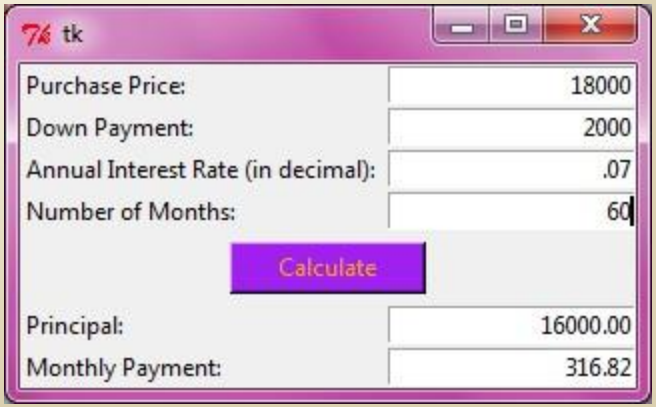
Event-Driven Programming



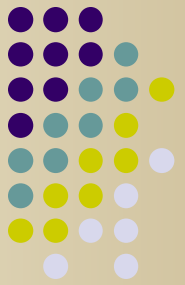
- Modern GUIs are notification-based
- Each program registers a handler function for events it is interested in. E.g., keypress, mouse click, mouse move...
- Handlers also are called ***callback*** functions

Graphical User Interface (GUI)

- User interface design is separate from computational design
- Want to create a GUI for the program from Lecture 4 that computes a monthly loan payment.
- Function to compute the monthly loan payment is the same.



Python GUIs



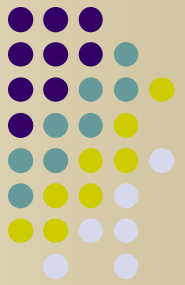
- There are several different libraries for creating a GUI in Python.
- The Tkinter library is built into the Python system and is an interface to the Tcl/Tk language. It needs to be imported.

```
from Tkinter import *
```

- The main window is created and started using

```
topWindow = Tk()
topWindow.mainloop()
```

Widgets



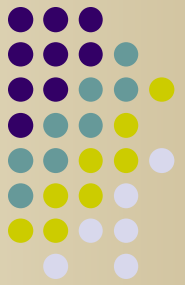
- GUI elements are called ***widgets***.
- When widgets are created, they are given a parent GUI element and any options. The options are passed using keywords.
- Common options include text, foreground(fg), background (bg), width, height, padx, pady

Labels

- A Label is a widget that holds text.

```
lbl = Label(root, text="Hello!")
```

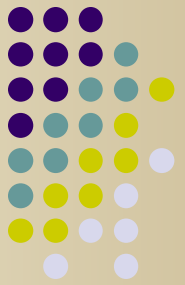
Entries



- An Entry is used for data entry. One of its options is justification of the text displayed in the entry. The text itself must be inserted starting at a particular index.

```
inputEntry = Entry(root, justify=RIGHT)
inputEntry.insert(0, "0")
```

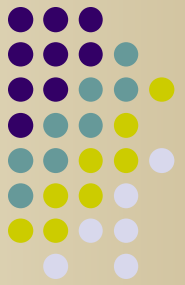
Buttons



- A Button has a command option that is a function that is executed when the button is clicked.

```
submitBtn = Button(root,
                   text='Calculate',
                   command=submitHandler,
                   bg='purple',
                   fg='orange', padx=20)
```

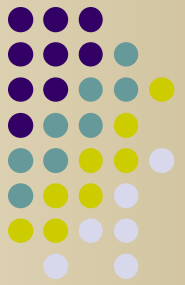
Layout Management



- Widgets are arranged using a ***layout manager***. Python has several, but we will be using the ***grid*** manager, since it is the easiest to understand.
- As the name implies, the grid manager lays out widgets in a grid. A row and a column index is given. E.g.

```
submitBtn.grid(row=0, column=0)
  # (0,0) is upper-left corner
```

Layout Management



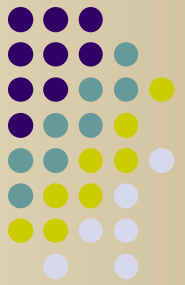
- The size of the rows and columns depend on the largest widget being placed in the row/column.
- When a widget is smaller than its cell, the sticky option can be used to justify it.

```
lbl.grid(row=rowNum, column=0, sticky='W')
```

- Widgets also can span rows/columns

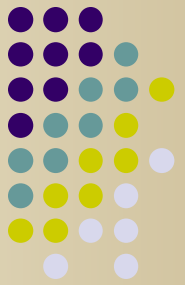
```
submitBtn.grid(row=rowNum, column=0,
                    columnspan=2)
```

makeForm Function



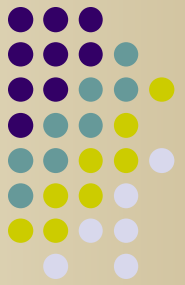
- The function **`makeForm`** in **`lecture15.py`** will create the GUI for the program. Each row consists of a Label (in column 0) and an Entry (in column 1). The Label texts are the field names that come from the **`inputFields`** and **`outputFields`** lists.
- A dictionary is created with the field names as keys and the Entry widgets as the corresponding values, so that the button handler can access the Entry widgets.

makeForm Design, Part 1



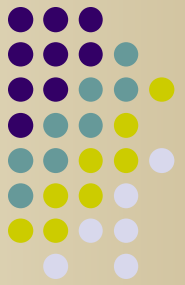
1. Start rowNum at 0 and create an empty dictionary
2. For each field name in the input field list
   a. **Create a Label with the field name**
   b. **Put the Label in grid location (rowNum, 0), left justified**
   c. **Create an Entry with right justified text**
   d. **Set the Entry text to "0"**
   e. **Put the Entry in grid location (rowNum, 1)**
   f. **Put <Label, Entry> pair into the dictionary**
   g. **Increment rowNum**

makeForm Design, Part 2



3. **Create a Button connected to a handler**
4. Put the Button into the grid (e.g. location (rowNum, 0)) updating rowNum as appropriate
5. For each field name in the output field list
   a. Do the same as for the input fields
6. Return the dictionary

In-class Exercise



- Implement the body of the first for-loop in the **`makeForm`** function. That is, for each row, create a Label and an Entry, and place them into the grid of the root widget.
- Create a Button with handler function **`submitHandler`** and place it between the input and output fields.

In-class Exercise

- Run the program. It should create a window, but since there's no handler, it does not compute anything.
- Experiment with the colors and placement of the button. See if you can put the button to the right of the entries in the middle of the column.