CS 112: Intro to Comp Prog

- Tkinter
- Layout Managers: place, pack, grid
- Custom Frames
- Widgets In-depth
- StringVar
- tkFont
- Upcoming
To use Tkinter Widgets (components: buttons, labels, etc). You must import it

```python
from Tkinter import *
```

Tk is the top level widget of Tkinter which represents the main window of an application. The simplest Tkinter application:

```python
from Tkinter import *
Tk().mainloop()
```

----------OR----------

```python
from Tkinter import *
root = Tk()
root.mainloop()
```
A widget is a fundamental unit of Python GUIs, anything you see on the GUI is in fact a widget:

**Image slightly modified from Dr. Heishman's CS 112 Slides**
Let's add four buttons to our window:

```python
from Tkinter import *

root = Tk()
btn_1 = Button(root, text = "Button 1")
btn_2 = Button(root, text = "Button 2")
btn_3 = Button(root, text = "Button 3")
btn_4 = Button(root, text = "Button 4")
root.mainloop()
```

**NOTHING HAPPENED?**

- Python didn't know how to place your widgets on the root window. We need to use a Layout Manager.
A Layout manager is a system that determines how to place and size widgets/components.

Python provides three:

- **place** - Allows programmer to state exactly the location and size of the widgets
- **pack** - places components in order (a stacking-type approach)
- **grid** - places components in a grid location (table-like approach)

Each has their own benefits and pitfalls.
• Inside the parenthesis we need to specify at least the location of the widget. Example below of placing btn_1 at the top-left corner

```python
btn_1.place(x = 0, y= 0)
```

---or---

```python
btn_1.place(relx = 0, rely = 0)
```

• `x` and `y` refer to absolute coordinates in pixel dimension (0,0) is the top-left corner and increase by going to the right (x-direction) or down (y-direction)

• `relx` and `rely` refer to relative coordinates based on the component it is placed in. Must be a value in [0.0, 1.0] (ex. 0.5 is the center of the parent)

• `width` and `height` are another common arguments as well, which override the default size (size is in pixels)

• Using this manager it is possible to have widgets overlap
• **anchor** is another option, which specifies which part of the widget will be located at the specified coordinates, default is 'nw' (north-west...top-left), possibilities: 'n', 'ne', 'e', 'se', 's', 'sw', 'w', 'nw', 'c'

• Example of placing the 4 buttons from earlier using a variety of arguments:

```python
from Tkinter import *

root = Tk()
btn_1 = Button(root, text = "Button 1")
btn_2 = Button(root, text = "Button 2")
btn_3 = Button(root, text = "Button 3")
btn_4 = Button(root, text = "Button 4")
btn_1.place(x = 0, y = 0, width = 150)
btn_2.place(relx = 0.5, rely = 0.5, height = 50, anchor = 'c')
btn_3.place(x=20,y=15)
btn_4.place(x = 30, rely = 0.5, anchor = 'se')
root.mainloop()
```
pack()

• pack() doesn't need any arguments for it places the widget, as the default values will place widgets vertically

```python
btn_1.pack()
```

• `side` refers to how the components will be arranged: vertically (keywords **TOP** *(default)* or **BOTTOM**) or horizontally (keywords **LEFT** or **RIGHT**)...top and left are typically the ones to use

• `fill` refers to resizing...should the component resize horizontally (**X**) or vertically (**Y**) to fill the parent widget (**X, Y, NONE** *(default)*, and **BOTH** are valid values)

• There are other argument values, but these two are the most common
● Example of placing the 4 buttons from earlier using a variety of arguments:

```python
from Tkinter import *

root = Tk()
btn_1 = Button(root, text = "Button 1")
btn_2 = Button(root, text = "Button 2")
btn_3 = Button(root, text = "Button 3")
btn_4 = Button(root, text = "Button 4")
btn_1.pack(fill = X, side = BOTTOM)
btn_2.pack(fill = X, side = TOP)
btn_3.pack(fill = Y, side = RIGHT)
btn_4.pack(fill = Y, side = LEFT)
root.mainloop()
```
grid()

- grid() doesn't need any arguments for it places the widget, as the default values will place widgets vertically (column = 0...row increments)

btn_1.grid()

- row and column refer to which grid location it belongs in (0,0) is the top-left spot with the values increasing to the right and down

- columnspan and rowspan refer to the number of columns or rows the widget will occupy

- sticky defines what side the component should stick to (if N it will be locked to the top of the cell) (N,S,E,W are valid options...you can also set the value as a combination: for example N+S means it will expand vertically to fill the cell

- There are other argument values, but these two sets are the most common

- NEVER use grid() and pack() in the same parent!!! It will freeze python!
Example of placing the 4 buttons from earlier using a variety of arguments:

```
from Tkinter import *

root = Tk()
btn_1 = Button(root, text = "Button 1")
btn_2 = Button(root, text = "Button 2")
btn_3 = Button(root, text = "Button 3")
btn_4 = Button(root, text = "Button 4")
btn_1.grid(row=0, column=0, rowspan = 2,sticky = N+S)
btn_2.grid(row=0,column=1)
btn_3.grid(row=1,column=1)
btn_4.grid(row=2, column=0, columnspan=2,sticky = E+W)
root.mainloop()
```
A frame is a specialized type of widget, which can contain other widgets (e.g., you can place Buttons, Labels, etc inside of it). Since it is also a widget, you can place Frames within Frames.

We can write our own Frame(s) that can be used multiple times, by using classes, Example:

```python
from Tkinter import *

class MyFrame(Frame):
    def __init__(self, root, name="Test"):
        Frame.__init__(self, root)
        self.btn_1 = Button(self, text = name+" 1")
        self.btn_2 = Button(self, text = name+" 2")
        self.btn_3 = Button(self, text = name+" 3")
        self.btn_4 = Button(self, text = name+" 4")
        self.btn_1.grid(row=0, column=0, rowspan = 2,sticky = N+S)
        self.btn_2.grid(row=0,column=1)
        self.btn_3.grid(row=1,column=1)
        self.btn_4.grid(row=2, column=0, columnspan=2,sticky = E+W)

root = Tk()
f = MyFrame(root, "One")
f2 = MyFrame(root, "Two")
f3 = MyFrame(root, "Three")
f4 = MyFrame(root, "Four")
f.grid(row = 0, column = 0)
f2.grid(row = 0, column = 1)
f3.grid(row = 1, column = 0)
f4.grid(row = 1, column = 1)
root.mainloop()
```
Widgets In-depth

- Upon construction of widgets we can pass in arguments that can set the size (so long as they are not using the place layout manager), text, or command depending on whether or not an event is being used, common arguments for some widgets are listed below:

  - **Button** -
    - **text** - set the text of the button
    - **command** - function name to be called, when button is pressed
  - **Label** -
    - **text** - set the text of the label
  - **Entry** -
    - (mainly use the common arguments below...particular importance on textvariable)
  - **Common arguments for all of the above:**
    - **background** - set the back color using strings..."black","blue","red",...
    - **font** - the size, family, bolded, ...font for the text (tkFont)
    - **foreground** - set the text color using strings..."black","blue","red",...
    - **textvariable** - associate a Tkinter StringVar variable to the text
Example of using a separate Frame class, that uses buttons, labels, and entries:

```python
from Tkinter import *

class MyFrame(Frame):
    def __init__(self, root):
        Frame.__init__(self, root)
        self.data = StringVar(self, "Before")
        self.setup_widgets()

    def setup_widgets(self):
        self.btn = Button(self, text = "My Button",
                           background = "green", foreground = "white",
                           command = self.press_btn)
        self.lbl = Label(self,foreground="blue",text = "Blah")
        self.entry = Entry(self, textvariable = self.data)
        self.lbl.grid(row=0,column=0)
        self.entry.grid(row=0,column=1)
        self.btn.grid(row=1,column=0,columnspan=2,sticky=W+E)

    def press_btn(self):
        self.data.set("After")

root = Tk()
F = MyFrame(root)
F.pack()
root.mainloop()
```
StringVar

- StringVar is a class that stores a string that can be accessed through get and set methods.
- It's constructor takes the parent (typically `self`) and an initial value:
  - `StringVar(self, "text")`
- When the `textvariable =` is used, the variable becomes associated with the text of the widget. If the StringVar is modified (set with new text, then all widgets that are associated with the variable, then their text will also be updated (The Entry in the last example)
- `get()` will return the current string value of the StringVar variable, can be used to get what the user changes the Entry to.
- `set("new text")` will updated and set the value to what is passed as an argument
from Tkinter import *

class MyFrame(Frame):
    def __init__(self,root):
        Frame.__init__(self,root)
        self.data = StringVar(self,"Before")
        self.setup_widgets()

    def setup_widgets(self):
        self.btn = Button(self, text = "Funky Button!",
        background = "green", foreground = "white",
        command = self.press_btn)
        self.lbl_1 = Label(self,foreground="blue",text = "Blah",
        textvariable = self.data)
        self.lbl_2 = Label(self,foreground="red",text = "Wha?",
        textvariable = self.data)
        self.lbl_3 = Label(self,foreground="black",text = "Snap",
        textvariable = self.data)
        self.lbl_1.grid(row=0,column=0)
        self.lbl_2.grid(row=0,column=1)
        self.lbl_3.grid(row=0,column=2)
        self.btn.grid(row=1,column=0,columnspan=3,sticky=W+E)

    def press_btn(self):
        self.data.set("After")

root = Tk()
f = MyFrame(root)
f.pack()
root.mainloop()
In order to use different fonts in python, you need to import tkFont

```
from tkFont import *
```

Then you when you construct the font you can set several arguments:

- **family** -- font 'family', e.g. Courier, Times, Helvetica
- **size** -- font size in points
- **weight** -- font thickness: NORMAL, BOLD
- **slant** -- font slant: ROMAN, ITALIC
- **underline** -- font underlinging: false (0), true (1)
- **overstrike** -- font strikeout: false (0), true (1)

Example of using fonts on next slide:
from Tkinter import *
from tkFont import *

class MyFrame(Frame):
    def __init__(self, root):
        Frame.__init__(self, root)
        self.data = StringVar(self, "My Text")
        self.setup_widgets()

    def setup_widgets(self):
        self.font1 = tkFont.Font(self, family="Courier", size = 20, underline=1)
        self.font2 = tkFont.Font(self, weight=BOLD, slant=ITALIC)
        self.font3 = tkFont.Font(self, overstrike = 1)

        self.lbl_1 = Label(self, font = self.font1, foreground="blue", textvariable = self.data)
        self.lbl_2 = Label(self, font = self.font2, foreground="red", textvariable = self.data)
        self.lbl_3 = Label(self, font = self.font3, foreground="black", textvariable = self.data)

        self.lbl_1.pack(fill=X)
        self.lbl_2.pack(fill=X)
        self.lbl_3.pack(fill=X)

root = Tk()
f = MyFrame(root)
f.pack()
root.mainloop()
The Text Widget is a bit different from the other widgets we have seen. There is no explicit text field we can set or get from. But we do have methods to get text, delete text, and insert text at certain locations in our text widget:

- `get(<<index1>>, <<index2>>)`
- `insert(<<index>>, <<string>>)`
- `delete(<<index1>>, <<index2>>)`

- `get` returns the string that is between `index1` (included) to `index2` (not-included)

- `insert` puts the specified string BEFORE index

- `delete` removes the characters between `index1` (included) to `index2` (not-included)

The indexes are specified as a string in the following format:

- "<<line>>.<<character>>" - lines start at 1, but characters start at 0
- index: "1.0" is the very first character in the Text widget
- END is a special index meaning one past the end of all text
- index: "1.end" meaning one past the end of line 1
from Tkinter import *
from tkFont import *

class MyFrame(Frame):
    def __init__(self, root):
        Frame.__init__(self, root)
        self.data1 = StringVar(self, "Default Text")
        self.data2 = StringVar(self, "Default Text")
        self.data3 = StringVar(self, "Default Text")
        self.setup_widgets()

    def setup_widgets(self):
        self.text = Text(self, width=30, height=10)
        self.lbl_1 = Label(self, textvariable=self.data1)
        self.lbl_2 = Label(self, textvariable=self.data2)
        self.lbl_3 = Label(self, textvariable=self.data3)
        self.btn = Button(self, text="Click ME!", command=self.click_me)
        self.text.grid(row=0, column=0, rowspan=3)
        self.lbl_1.grid(row=0, column=1)
        self.lbl_2.grid(row=1, column=1)
        self.lbl_3.grid(row=2, column=1)
        self.btn.grid(row=3, column=0, columnspan=2)

    def click_me(self):
        string1 = self.text.get("1.0","1.end")
        string2 = self.text.get("2.0",END)
        string3 = self.text.get("3.1", "3.3")
        self.text.delete("3.0","3.end")
        self.text.insert(END, "My New Text")
        self.data1.set(string1)
        self.data2.set(string2)
        self.data3.set(string3)

root = Tk()
f = MyFrame(root)
f.pack()
root.mainloop()
Other Widgets

Other Widgets are also available to use, but these are left to you to look up and use. Good resources for learning about these widgets:

- **Tkinter Reference**
- **Tkinter Module – JavaDoc style reference**
- **Introduction to Tkinter (Tkinter book)**
GUI Example - Calculator

Design a simple Calculator that can add, subtract, divide, and multiply [do not need sequential operations...i.e. 2+2+2= will give 6...have it only do one operation at a time...you must enter 2+2=+2= (to get 6)]
from Tkinter import *

class AppFrame(Frame):
    def __init__(self, root):
        Frame.__init__(self, root)
        self.setup_calc()
        self.setup_widgets()

    def setup_calc(self):
        prev_val = 0
        prev_op = ""
        self.data = StringVar(self, "0")

    def setup_widgets(self):
        # Define Widgets
        self.entry = Entry(self, textvariable = self.data)

        self.btn_1 = Button(self, text="1", command = self.click_1)
        self.btn_2 = Button(self, text="2", command = self.click_2)
        self.btn_3 = Button(self, text="3", command = self.click_3)
        self.btn_4 = Button(self, text="4", command = self.click_4)
        self.btn_5 = Button(self, text="5", command = self.click_5)
        self.btn_6 = Button(self, text="6", command = self.click_6)
        self.btn_7 = Button(self, text="7", command = self.click_7)
        self.btn_8 = Button(self, text="8", command = self.click_8)
        self.btn_9 = Button(self, text="9", command = self.click_9)
        self.btn_0 = Button(self, text="0", command = self.click_0)
        self.btn_dec = Button(self, text=".", command = self.click_dec)
        self.btn_plus = Button(self, text="+", command = self.click_add)
        self.btn_sub = Button(self, text="-", command = self.click_sub)
        self.btn_mult = Button(self, text="*", command = self.click_mult)
        self.btn_div = Button(self, text="/", command = self.click_div)
        self.btn_clear = Button(self, text="C", command = self.click_clear)
        self.btn_equals = Button(self, text="=", command = self.click_equals)
# Place widgets
self.entry.grid(row=0,column=0,columnspan = 4,sticky = W+E)
self.btn_7.grid(row=1,column=0,sticky = W+E)
self.btn_8.grid(row=1,column=1,sticky = W+E)
self.btn_9.grid(row=1,column=2,sticky = W+E)
self.btn_div.grid(row=1,column=3,sticky = W+E)
self.btn_4.grid(row=2,column=0,sticky = W+E)
self.btn_5.grid(row=2,column=1,sticky = W+E)
self.btn_6.grid(row=2,column=2,sticky = W+E)
self.btn_mult.grid(row=2,column=3,sticky = W+E)
self.btn_1.grid(row=3,column=0,sticky = W+E)
self.btn_2.grid(row=3,column=1,sticky = W+E)
self.btn_3.grid(row=3,column=2,sticky = W+E)
self.btn_sub.grid(row=3,column=3,sticky = W+E)
self.btn_0.grid(row=4,column=0,columnspan=2,sticky = W+E)
self.btn_dec.grid(row=4,column=2,sticky = W+E)
self.btn_plus.grid(row=4,column=3,sticky = W+E)
self.btn_clear.grid(row=5,column=0,columnspan=2,sticky = W+E)
self.btn_equals.grid(row=5,column=2,columnspan=2,sticky = W+E)

def click_0(self): self.click_number("0")
def click_1(self): self.click_number("1")
def click_2(self): self.click_number("2")
def click_3(self): self.click_number("3")
def click_4(self): self.click_number("4")
def click_5(self): self.click_number("5")
def click_6(self): self.click_number("6")
def click_7(self): self.click_number("7")
def click_8(self): self.click_number("8")
def click_9(self): self.click_number("9")
def click_number(self, str_num):
    if(self.data.get() == "0"): 
        self.data.set(str_num)
    else:
        self.data.set(self.data.get()+str_num)
def click_add(self): self.click_op("+")
def click_sub(self): self.click_op("-")
def click_mult(self): self.click_op("*")
def click_div(self): self.click_op("/")

def click_op(self, str_op):
    self.prev_val = float(self.data.get())
    self.data.set("0")
    self.prev_op = str_op

def click_dec(self):
    if "." not in self.data.get():
        self.data.set(self.data.get() + ".")

def click_clear(self):
    prev_val = 0
    self.data.set("0")

def click_equals(self):
    if self.prev_op == ": return
    cur_val = float(self.data.get())
    if self.prev_op == "+":
        self.data.set(str(self.prev_val + cur_val))
    if self.prev_op == "-":
        self.data.set(str(self.prev_val - cur_val))
    if self.prev_op == "*":
        self.data.set(str(self.prev_val * cur_val))
    if self.prev_op == "/":
        self.data.set(str(self.prev_val / cur_val))

root = Tk()
f = AppFrame(root)
f.pack()
root.mainloop()