Tkinter for Python

Toolkit for Interfaces
GUI programming

- **GUI** (pronounced “gooey”) stands for Graphical User Interface
- In GUI programming, the function of the “main” method, if present at all, is to create the graphical user interface
- Thereafter, everything that happens is controlled from the interface
- When the GUI is asked to do something (for example, by the user clicking a button), the GUI can call a function with no parameters
Setup

• Begin with this `import` statement:
  
  ```python
  from tkinter import *
  ```

  • Note: In earlier versions of Python, this module was called `Tkinter`, not `tkinter`

• Then create an object of type `Tk`:
  ```python
  top = Tk()
  ```

  • This is the top-level `window` of your GUI program

  • You can use any name for it; in these slides I use “`top`”

• Define the functions you are going to use

• Create `widgets` (graphical elements) and add them to the window

• Run the program by calling `mainloop()`
from tkinter import *

top = Tk()

def more():
    l = Label(top, text="Ouch!")  # create label
    l.pack()  # add to window

b = Button(top, text="Don't click me!", command=more)
b.pack()

mainloop()
Rearranging the example

- In Python, the code is executed as it is encountered
  - In the first example, the more function had to be defined before it could be referred to
  - Encapsulating code in methods allows it to be arranged in any order
- `from tkinter import *`

```python
from tkinter import *

top = Tk()

def main():
    b = Button(top, text="Don't click me!", command=more)
    b.pack()
    mainloop()

def more():
    Label(top, text="Ouch!").pack()

main()
```
Building a GUI

• Building a GUI requires:
  • Defining a number of widgets (easy)
  • Defining functions for the widgets to call (standard Python programming)
    • Don’t use `print` statements, though!
  • Arranging the widgets in the window (can be difficult to get what you want)
• All the widgets, and the methods to arrange them, take a large number of parameters
  • Use `named` parameters--don’t try to memorize the order!
• **Example:** `Button(top, text="Don't click me!", command=more)`
Widgets I

- Here are some typical widgets, with typical parameters

  - `but = Button(top, text=string, command=function)`
  - `lab = Label(top, text=string)`
  - `chk = Checkbutton(top, text=string)`
  - `ent = Entry(top, width=n)`
  - `txt = Text(top, width=num_characters, height=num_lines)`
Important advice

• Build your GUI a little bit at a time, and run it after every little change!
  • Why? You don’t get runtime error messages!
  • Here’s what you get for a runtime error:

When you see this, it’s time to examine carefully the last code you added
Making widgets active

• With many of the widgets, you can add the parameter `command=function`

• Some widgets, such as buttons and menu items, should do something when clicked
  • And the change should be visible to the user!!!

• Most widgets, such as text entry areas and checkboxes, should *not* do anything
  • Instead, the program should ask the widget for its value, if and when that value is needed
  • For example, both `Entry` and `Text` have a `get` method to return the text currently in them
Second example

```python
def main():
    global lab, chk, chkvar, ent, txt
    Button(top, text="Here is my Button",
           command=button).pack()
    lab = Label(top, text="I am a Label", width=20)
    lab.pack()
    chkvar = IntVar()
    chk = Checkbutton(top, text="This is a Checkbutton",
                      variable=chkvar)
    chk.pack()
    ent = Entry(top, width=25)
    ent.pack()
    txt = Text(top, width=25, height=3)
    txt.pack()
    mainloop()
```
Second example

```python
def main():
    global lab, chk, chkvar, ent, txt
    Button(top, text="Here is my Button", command=button).pack()
    lab = Label(top, text="I am a Label", width=20)
    lab.pack()
    chkvar = IntVar()
    chk = Checkbutton(top, text="This is a Checkbutton", variable=chkvar)
    chk.pack()
    ent = Entry(top, width=25, text="This is an Entry")
    ent.pack()
    txt = Text(top, width=25, height=3)
    txt.pack()
mainloop()
```
Explanations I

- `global lab, chk, chkvar, ent, txt`
  - These widgets are made global so that I can refer to them outside of the `main` method
  - Of course, if they are not in a method, I don’t need to do this

- `Button(top, text="Here is my Button",\n  command=button).pack()`
  - This `Button`, when clicked, will call my badly-named function `button`
  - I will never need to refer to this `Button`, so I don’t bother assigning it to a variable
Explanations II

- `lab = Label(top, text="I am a Label", width=20)`
  `lab.pack()`
  - For a button I just said `Button(...) . pack()`, because I didn’t need to ever refer to the button again
  - However, the `pack()` method returns `None`, so `lab = Label(...).pack()` would assign `None` to `lab`
  - Therefore, I had to pack the label on a separate line

- `chkvar = IntVar()`
  `chk = Checkbutton(top, text="This is a Checkbutton",
  variable=chkvar)`
  - This is how you find out whether a `Checkbutton` has been checked: `chkvar.get()`
  - The result is (by default) `1` if checked, `0` if not checked
Explanations III

- `lab = Label(top, text="I am a Label", width=20)
  lab.pack()`

  - For a button I just said `Button(...).pack()`, because I didn’t need to ever refer to the button again

  - However, the `pack()` method returns None, so `lab = Label(...).pack()` would assign None to `lab`

  - Therefore, I had to pack the label on a separate line

- `chkvar = IntVar()
  chk = Checkbutton(top, text="This is a Checkbutton",
                   variable=chkvar)`

  - This is how you find out whether a `Checkbutton` has been checked: `chkvar.get()`

  - The result is (by default) 1 if checked, 0 if not checked
Explanations IV

- \texttt{ent = Entry(top, width=25)}
  \texttt{ent.pack()}

- \texttt{txt = Text(top, width=25, height=3)}
  \texttt{txt.pack()}

- To retrieve text from an \texttt{Entry}, use:
  \texttt{ent.get()}

- To retrieve text from a \texttt{Text}, use:
  \texttt{txt.get(1.0, END)}
A **Frame** is a widget whose purpose is to hold other widgets

- All but the very simplest GUIs use frames, and often frames within frames
- The program arranges the frames within the window, and arranges widgets within the frames
- There are three functions for inserting widgets into frames (or into windows): **pack**, **grid**, and **place**

- I recommend against using **place**
- I *strongly* recommend not using a mix of functions in any given frame (or window); they don’t get along well
pack

- **pack** has a parameter `side` which can be set to one of the strings `left`, `right`, `up`, or `down`.
- **pack** has parameters `padx` and `pady` that can be set to give padding (measured in pixels) around the widget.
- `top['bg'] = 'light gray'
  ```python
  Button(top, text="Left 1").pack(side='left')
  Button(top, text="Left 2").pack(side='left')
  Button(top, text="Right 1").pack(side='right')
  Button(top, text="Right 2").pack(side='right', padx=10)
  Button(top, text="Top 1").pack(side='top')
  Button(top, text="Top 2").pack(side='top')
  Button(top, text="Bottom 1").pack(side='bottom')
  Button(top, text="Bottom 2").pack(side='bottom', pady=10)
  ```
  mainloop()
pack

- **pack** has a parameter side which can be set to one of the strings: 'left', 'right', 'up', or 'down'.
- **pack** has parameters **padx** and **pady** that can be set to give padding (measured in pixels) around the widget.
- ```python
   top['bg'] = 'light gray'
   Button(top, text="Left 1").pack(side='left')
   Button(top, text="Left 2").pack(side='left', padx=10)
   Button(top, text="Right 1").pack(side='right')
   Button(top, text="Right 2").pack(side='right', padx=10)
   Button(top, text="Top 1").pack(side='top')
   Button(top, text="Top 2").pack(side='top')
   Button(top, text="Bottom 1").pack(side='bottom')
   Button(top, text="Bottom 2").pack(side='bottom', pady=10)
   mainloop()
```
grid

- The `grid` function has parameters `row` and `column`, as well as `padx` and `pady`.

- `top['bg'] = 'light gray'
  Button(top, text="One").grid(row=0, column=0)
  Button(top, text="Two").grid(row=0, column=1, pady=10)
  Button(top, text="Three").grid(row=0, column=2)
  Button(top, text="Four").grid(row=1, column=2)
  Button(top, text="Five").grid(row=1, column=1)
  Button(top, text="Six").grid(row=1, column=0)
mainloop()
The `grid` function has parameters `row` and `column`, as well as `padx` and `pady`.

```python
top['bg'] = 'light gray'
Button(top, text="One").grid(row=0, column=0)
Button(top, text="Two").grid(row=0, column=1, pady=10)
Button(top, text="Three").grid(row=0, column=2)
Button(top, text="Four").grid(row=1, column=2)
Button(top, text="Five").grid(row=1, column=1)
Button(top, text="Six").grid(row=1, column=0)
mainloop()
```
Frames are used to hold and organize other widgets

```python
• top['bg'] = 'light gray'
frame1 = Frame(top, bg = '#FFCCCC')
frame1.pack(side=LEFT)
Button(frame1, text="One", fg='red').grid(row=0, column=0)
Button(frame1, text="Two").grid(row=0, column=1, pady=10)
Button(frame1, text="Three").grid(row=0, column=2)
frame2 = Frame(top, bg='cyan')
frame2.pack(side='right')
Button(frame2, text="Big Fat Four").pack(side=TOP)
Button(frame2, text="Five").pack(side='top')
Button(frame2, text="Six").pack(side='top', fill=BOTH)
mainloop()
```
Frames are used to hold and organize other widgets

```python
top['bg'] = 'light gray'
frame1 = Frame(top, bg = '#FFCCCC')
frame1.pack(side=LEFT)
Button(frame1, text="One", fg='red').grid(row=0, column=0)
Button(frame1, text="Two").grid(row=0, column=1, pady=10)
Button(frame1, text="Three").grid(row=0, column=2)
frame2 = Frame(top, bg='cyan')
frame2.pack(side='right')
Button(frame2, text="Big Fat Four").pack(side=TOP)
Button(frame2, text="Five").pack(side='top')
Button(frame2, text="Six").pack(side='top', fill=BOTH)
mainloop()
```
Explanations I

- `top['bg'] = 'light gray'`
  - Many widgets have `fg` (foreground) and `bg` (background) attributes
  - The available color names vary from system to system, but you can count on having 'black', 'white', 'red', 'yellow', 'green', 'blue', 'cyan', and 'magenta'

- `frame1 = Frame(top, bg = '#FFCCCC')`
  - Color names can also be given as a hex string

- `Button(frame1, text="One", fg='red')`...
  - Setting the foreground usually means setting the color of text
  - On a Mac, setting the background color is legal but is ignored
Explanations II

- `Button(frame1, text="Two").grid(..., pady=10)`
  - This asks for padding above and below button “two”
  - The window background is light gray, but the frame background is pink, so the padding is pink
  - The other buttons in the row also get the padding

- `Button(frame2, text="Big Fat Four").pack(side=TOP)`
  - Some things can be represented in two ways, such as ‘top’ and TOP
  - The “four” button is wider than the others, so the column is made that wide
  - Since the “five” button isn’t as wide, we see the background on both sides
  - We ask the “six” button to fill the available space, BOTH in x and in y
Problems

• It’s not hard to build a GUI, but you may get little or no help with errors
  • You can get a blank window, or no window at all
  • *Really* do it a little at a time, and test after every step!
• Did you forget to **pack** or **grid** your widget?
• *'NoneType' object does not support item assignment* -- Did you do `w = SomeWidget(...).pack()` and set `w` to **None**?
• Did you forget the **mainloop()**?
• Did you try to **print** when running a GUI?
• *'str' object has no attribute 'items'* -- Did you forget to put **text=** before a string?
References

• We have barely scratched the surface of what Tkinter can do.
• There are 15 kinds of widgets, and each has lots of attributes.
• [https://www.tutorialspoint.com/python/python_gui_programming.htm](https://www.tutorialspoint.com/python/python_gui_programming.htm) is a good reference, but...
  • It can be really slow (because of Flash).
  • Some widget descriptions appear to be copied and pasted from other widget descriptions, and not edited.
The End

def quit():
    top.destroy()
    exit()

quitButton = Button(top, text="Quit",
                   command=quit).pack()