Python GUI
Ada GUI

Python: Import GUI Toolkit Module

- Tkinter: included in standard Python distribution
  - Statement: **import Tkinter**
- wxPython can also be used but is not part of the standard Python distribution and thus has to be downloaded separately.

Python: Create a Class

- When use Tkinter, inheriting from Tkinter.Tk
  - Tkinter.Tk is the base class for standard GUI windows
  - Statement: **class simpleapp_tk(Tkinter.Tk)**

Python: Constructors

- Declaring a constructor: **def __init__(self,parent)**
- Need to pass ‘parent’ into the constructor
  - GUI is a hiearchy of objects (e.g., window, tab, pane, button)
  - Each GUI object has a parent, which usually is the container in which the element resides
  - Can keep reference to the parent when building a GUI component (good practice): **self.parent = parent**

Python: Initialize

- Generally, code is written that creates all of the GUI elements (buttons, text fields, etc.) in an **initialize()** method. This method should not include any logic, just creation statements and initialization statements of GUI elements (or widgets)
- Declaring an initialization method:

```
Def initialize(self):
        …
        …
```

- Calling an initialization method (can call this method in the constructor):

```
def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()
```

Python: Main (Method)

- We need to create a **main** method that is executed when the program is run from the command line.
- When using Tkinter, we need to instanciate the class from the main method with the statement below. There is no parent because it is the main window (the first GUI element).

app=simpleapp_tk(None)

- We can give it a title using the command: **app.title('Some Title')**

Python: Infinite Loop (keep program running)

- We need to keep a GUI program running, and so we create an infinite loop with the following statement:

  app.mainloop()

- Keeping the program in an infinite loop allows it to wait for events (e.g., clicking buttons, pressing keys, etc.)

Python: Layout

- We can declare a layout that allows the programmer to space GUI elements within a container/window.
- These widgets can be added next to each other side by side, horizontally, vertically, in a grid, etc.
- You can declare a layout on each window, pane, tab, dialog, etc.
- In this example, we are declaring a grid layout with the statement below:

self.grid()

- In a grid layout, widgets are positioned by column/row

Python: Widgets (GUI Elements)

- We add widgets to our grid layout:
  - Create a text field widget (we are passing 'self' to the text widget because 'self' is a reference to the main window, which is the parent for this widget):

self.entry = Tkinter.Entry(self)

  - Adding the text field to the grid layout with the statement below.
    - 'self.entry' allows us to keep a reference to this widget in our class
    - 'stiky='EW'' allows use to dictate how the widget responds to a cell growing larger. For example, if the cell grows larger, we can have the widget that the cell contains to stick to edges of the cell (E = east (left), W = west (right), N = north (top), and S = south (bottom)).

self.entry.grid(column=0,row=0,stiky='EW')

Python: More Widgets (GUI Elements)

- Adding a button
  - Create a button widget (we are not keeping a reference to the button because we won't need to read or write a value to the button):

button = Tkinter.Button(self, text=u"Click me !")

  - Adding the button to the grid layout with the statement below.

button .grid(column=1,row=0)

Python: More Widgets (GUI Elements)

- Adding a label
  - Create a label widget:
    - We are adding a foreground and background color
    - We are adding a text alignment (anchor="w" forces the text to be aligned in the label)

label = Tkinter.Label(self, anchor="w",fg="white", bg="blue")

  - Adding the label to the grid layout with the statement below.
    - The columnspan attribute allows us to force this label to span across two cells (in this example, the label will span across below both the text field and the button

label.grid(column=0,row=1,columnspan=2,sticky='EW')

Python: Resizing

- We can allow resizing with the statement below.
  - This tells the layout manager to resize its columns and rows when the window is resized.
  - In our example, we are resizing only the first column.
  - The ‘weight’ parameter allows some columns resize more than others.

self.grid_columnconfigure(0,weight=1)

Python: Resizing Constraints

- We can restrict resizing of windows.
  - In this example, we restrict the window to allow resizing only horizontally.

self.resizeable(True,False)

Python: Event Handlers

- We can declare event handlers that get triggered based on some action.
  - Adding an event handler for the text field
    - First, bind an event handler to a specific widget
    - 'Return' is the event to catch (e.g., click)

```
self.entry = Tkinter.Entry(self)
self.entry.grid(column=0,row=0,sticky='EW')
self.entry.bind("<Return>", self.OnPressEnter)

button = Tkinter.Button(self,text=u"Click me !",
                        command=self.OnButtonClick)
button.grid(column=1,row=0)
```

    - Second, implement methods that handle the events:

```
def OnButtonClick(self):
    print "You clicked the button !"

def OnPressEnter(self,event):
    print "You pressed enter !"
```

Python: Changing Label Event

- We can change a label when an event is triggered.
  - Create a Tkinter variable (self.labelVariable = Tkinter.StringVar())
  - bind it to the widget (textvariable=self.labelVariable)
  - use set() or get() to set or read its value (self.labelVariable.set("You clicked the button !"))
  - In Tkinter, each time you want to read or set values in a widgets (text field, label, checkbox, radio button, etc.), you have to create a Tkinter variable and bind it to the widget.

```
self.labelVariable = Tkinter.StringVar()
    label = Tkinter.Label(self,textvariable=self.labelVariable,
                          anchor="w",fg="white",bg="blue")
    label.grid(column=0,row=1,columnspan=2,sticky='EW')

    self.grid_columnconfigure(0,weight=1)
    self.resizable(True,False)

  def OnButtonClick(self):
    self.labelVariable.set("You clicked the button !")

  def OnPressEnter(self,event):
    self.labelVariable.set("You pressed enter !")
```

Python: Display a Value

- We can change a label when an event is triggered.
  - we have to create a Tkinter variable again, so that we can do self.entryVariable.get().

```
self.entryVariable = Tkinter.StringVar()
    self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
    self.entry.grid(column=0,row=0,sticky='EW')
    self.entry.bind("<Return>", self.OnPressEnter)
    self.entryVariable.set(u"Enter text here.")

    ...
    ...
    self.labelVariable.set(u"Hello !")
```

Ada: Programming Environment

- GNAT Ada GPL : development environment (IDE and compiler) for Ada
- **GtkAda:** GUI development environment for Ada

- **Download:** http://libre.adacore.com/download/configurations

Ada: Programming Environment

- GNAT Ada GPL : development environment (IDE and compiler) for Ada
- **GtkAda:** GUI development environment for Ada

- **Download:**
  http://libre.adacore.com/download/configurations

  - **GNAT Ada GPL 2014**
    - **Installs an IDE and compiler**
    - **Launch 'GPS' (this is an IDE), and create a project to create the source files under this project**
  - **GtkAda GPL 2014**
    - **Installs the GTK**
    - **GtkAda includes both Gtk and Glade for GUI development.**
    - **Navigate to the locally created directory (C:\GtkAda\share\examples\gtkada\testgtk) and run the executable testgtk.exe (the source code for this demo was uploaded to the recitation site)**
  - **Win32Ada GPL 2014 (only for Windows)**
    - **Gets added to the GNAT Ada GLP libraries**