An Introduction to Python Programming and GUI Design Using Tkinter

Bruno Dufour

McGill University SOCS

What is Python?

Python is a scripting laguage whis is:

- interpreted
- interactive
- object-oriented
- like pseudocode
- dynamically typed
- available for many platforms

Why is UI design important?

- User interfaces are what allows end users to interact with an application.
- A good UI will make an application intuitive and easy to use
- Excellent applications without good UI will be less popular than inferior ones with a good UI

What makes a good UI?

- Simple
- Intuitive
- Respects the commonly accepted conventions
- Visually organized
- Native look

Designing UIs

- Two levels:
  - Graphical: visual aspect
  - Events: Funcionality
- Basic blocks = *widgets*
- Making an application react to events = *binding*

Tkinter and UI design

Tkinter is a GUI toolkit for Python which is:

- Open-source
- Portable
- Layered: wrapper around Tk, which is implemented in C.

Competing GUI toolkits

- wxPython: 2^{nd} most popular. Good for complex UIs.
- JPython: access to the Swing library
- PyQt: Access to the well-known Qt library
- win32all.exe: Access to MFC from python (MS-Win only)
- WPY: MFC style, both also available for UNIX
- X11: limited to X Windows.

Why Tkinter?

- Layered design
- Accessibility
- Portability
- Availability

Other toolkits do not have all of these features at the same time.

Why NOT Tkinter?

Tkinter’s layered design has 2 major drawbacks:

- slows down applications
- Makes source more difficult to read

Show me what you can do. . .

A simple, “hello world”-like example in 4 lines:

```
from Tkinter import *
root = Tk( )
root.title('A simple application')
root.mainloop( )
```

Try running this from an interactive python session. . .

Tkinter widgets

Tkinter provides numerous widgets, among which:

- Button
- Checkbutton
- Entry
- Label
- Radiobutton
- . . .

The Canvas Widget

The canvas widget is one of the most powerful ones in the Tkinter library

- It provides drawing facilites through the creation of *items*
- Allows each item to have zero or more *tags*
- Allows each item to respond to *events*
- Supports output to PostScript format using a single call to postscript()

Note: items are not widgets!

The Standard Canvas Items

- Arc: arc, chord, pieslice
- Bitmap: builtin or read from an XBM file
- Image
- Line
- Oval: circle or ellipse
- Polygon
- Resctangle
- Text
- Window: places other widgets on the canvas

Canvas coordinates

- Origin = top-left corner
- 2 simlutaneous systems:
  - Canvas coordinate system
  - Window coordinate system (in events)
- Convert from one to the other using canvasx() and canvasy()

How to draw on a Canvas

- Drawing is done by creating new canvas items
- items are created using create_XYZ(coords, options), where XYZ is the name of an item type (eg: create_line(0, 0, 100, 200, fill='blue'))
- create_XYZ() returns an item ID (an integer)
- The item ID is used to access the item's configuration

Manipulating Items

- coords(item, x0, y0, x1, y1, . . .): modifies the item’s coordinates
- delete(item): removes “item”
- itemcget(item, option): returns the value of “option” for “item”
- itemconfigure(item, options): Modifies one or more options for all matching items.
- lift(item), lower(item): moves “item” to the top / bottom of item stack

Manipulating Items (cont.)

- move(item, dx, dy): moves an item by dx in the x direction and dy in the y direction
- scale(item, xscale, yscale, xoffset, yoffset): scales an item (x/yoffset are subtracted from the coordinates before scaling, then added back)

Finding Items

- find_above(item)
- find_all()
- find_below(item)
- find_closest(x, y)
- find_enclosed(x1, y1, x2, y2)
- find_overlapping(x1, y1, x2, y2)
- find_withtag(tag)

Canvas and Tags

Tags are a very powerful yet simple way to manipulate items:

- Each item can have any number of tags
- Many items can share a tag, which defines a group
- Tags and item IDs are interchangeable in all Canvas methods

Canvas and Tags (cont.)

Two special tags:

- CURRENT: automatically added to the item which is under the mouse cursor
- ALL: belongs to all items on the canvas

Manipulating Tags

- addtag_above(newtag, item)
- addtag_all(newtag)
- addtag_below(newtag, item)
- addtag_closest(newtag, x, y, halo=None, start=None)
- addtag_enclosed(newtag, x1, y1, x2, y2)
- addtag_overlapping(newtag, x1, y1, x2, y2)
- addtag_withtag(newtag, tag)
- dtag(item, tag)

Geometry Management

Geometry management consists of:

- Placing widgets in their *geometry masters*
- Sizing the widgets
- Negotiating between all widgets to properly adapt to changes.

Tkinter provides 3 ways to achieve this

Tkinter Geometry Managers

1. Pack Geometry manager
2. Grid
3. Place

1. Pack Geometry Manager

- quickest
- works by taking a portion of the space left in the master and attributing it to a widget
- More complex layouts require the use of *frames*

2. Grid Geometry Manager

- Makes it easy to produce complex grids of widgets
- Works in way which closely resembles HTML tables
- Widgets all have a particular cell, and may span multiple ones.

Note: Grid and Pack cannot be used simultaneously, as this results in an infinite negociation loop

3. Placer

- most precise
- Allows exact placement of widgets by
  - Exact coordinates and/or size
  - relative coordinates and/or size
- not commonly used, since it requires more efforts

Tkinter Specific Topics

- Specifying Colors
- Specifying Fonts
- Tkinter Variables

Specifying Colors

Tkinter provides 2 ways to describe colors:

- By Name: the standard X color names are guaranteed to be available (eg: “Blue”)
- By Color Strings: “#RGB”, “#RRGGBB”, “#RRRRGGGGBBBB”

Specifying Fonts

- Fonts can be specified using tuples in the form (*family*, *size*, [*option1*], [*option2*], . . .).
- eg: (’arial’, 10, ’bold’)
- Font Instances: instances of the tkFont.Font class
- eg: myfont = tkFont.Font(family="arial", size=20, weight=tkFont.BOLD)
- modifications can be done using the config() method

Tkinter Variables

- Created in order to have a way to respond to changes in the value of the variable (eg update a label when its text variable is updated)
- Provides the set() and get() methods for accessing the values
- Available Tkinter Variables: StringVar, IntVar, DoubleVar, BooleanVar

Event Handling

- Easy, convenient and flexible
- Allows callbacks to be associated with any event for any widget
- *Event descriptors* are used to identify events

Event Descriptors

- String Representation of Events
- Used for binding callbacks to events
- General Form: <Modifier - Type - Qualifier >
- Not all 3 sections are required

Event Types

The following are valid event types in Tkinter:

Keyboard events : KeyPress, KeyRelease

Mouse events : ButtonPress, ButtonRelease, Motion, Enter, Leave, MouseWheel

Window events : Visibility, Unmap, Map, Expose, FocusIn, FocusOut, Circulate, Colourmap, Gravity, Reparent, Property, Destroy, Activate, Deactivate

Event Qualifiers

Can be one of:

- Mouse button index (1 to 5)
- Keysym (eg: “BackSpace”)

Event Modifiers

- Control, Shift, Alt, Meta: Modifier keys
- B1 to B5: mouse buttons
- Double, Triple: repetition modifiers
- Any

More than one modifier can be specified at a time

Bind callbacks to events

3 methods:

- bind()
- bind_class()
- bind_all()

Callbacks and Events

- Tkinter always uses the most specific event when it has the choice
- Callbacks for the following 4 levels will be called in sequence: widget, Toplevel, class, application (in this order)
- Returning the string “break” at any of these levels will stop event propagation to lower levels.