Lecture #21 – Tk

- Tk

  Used for writing GUI applications with scripting syntax
  Superset of TCL (i.e. all TCL syntax is available)
  Also available as an extension to Perl (called Perl/Tk)

- Widgets (definition and creation)

  A **widget** is a graphical element in an application (i.e. a window, button, text box, etc.). Widgets are combined in a widget hierarchy (tree) to form an application.

  The root of the tree is the widget "."
  (i.e. the empty window that appears if you type "wish").

  Each widget has a name (ex. .menubar.edit.menu.quit)
  (i.e. its path from the root using "." As separators instead of "/" in the file system).

  Each parent in the widget hierarchy must be created before the child, and generally speaking, a widget is geometrically enclosed in its parent.

  For each widget type, there is a TCL command that creates widgets of that type. These commands take variable numbers of arguments.

  Example (Creating a button):

  ```
  button .b –text "Hello world" –command exit
  ```

  Note: The button is named ".b". It is a child of the root window.
  The –text option sets the label for the button, and the –command sets the action if the button is pressed.

  Once a widget is created, it is not immediately visible. To make a widget visible, it must be given to the geometry manager, and its parent widget must be visible.

  There are 3 geometry managers in Tk: "pack", "place", and "grid".
  The most widely used geometry manager is "pack".

  Example (packing a button):

  ```
  pack .b
  ```

- Manipulating widgets

  Once a widget is created, it can be referred to by its name.
  Some commands take the name of a widget as an argument.
  Also, the name of the widget is a command, which can be used to modify the widget.

  Example:

  ```
  button .b –text “hello” –command exit
  set label [ .b cget –text]
  .b configure –text “goodbye”
  .b invoke
  ```

  Note: cget is used to retrieve the current value of an option
  configure is used to change the value of an option
  invoke is used to “click” the button

- Some common widget options

-bg color	background color
-fg color	foreground color
-relief raised	set 3D aspect of the widget (also sunken, flat, ridge, groove)
-state disabled	greyed out
-state normal	active
-font fontname	font for the text (fontname created with the font command)

  Note: Colors can be names like “red” or hex strings “#RRGGBB”

- Geometry management

  Choice of three geometry managers – pack, place, and grid
  Pack is most common

- Pack

  The order widgets are packed in is very important (earlier widgets get preference).

  Pack allows you to control:

  Position in the window relative to the window or frame
  Size of widgets (either relative to other widgets or absolute)
  Spacing between widgets
  Position in the window’s or frame’s widget list

Pack options:

-side	left, right, **top**, bottom	
-fill	**none**, x, y, both	
-expand	1, **0**	
-anchor	n, ne, e, se, s, sw, w, nw, **center**	
-after	$otherwidget	
-before	$otherwidget	
-ipadx	amount	(increase size horizontally by amount x 2)
-ipady	amount	(increase size vertically by amount x 2)
-padx	amount	(padding on left and right)
-pady	amount	(padding on top and bottom)

Since default is “-side top”, packing widgets without options places them vertically in window.

- Button Widgets

There are several widgets that fall into the “button” category including buttons, labels, text entry boxes, checkbuttons, and radiobuttons.

Common options:

-text label	Value of the label
-textvariable var	Variable to hold the value of the label
-image img	Image to be display (created by the image command)
-command script	Action to be taken on activation
-variable var	Variable to hold the value of check or radio button
-value val	Value to be held by variable when check button selected
-onvalue val	Value for check button when selected
-offvalue val	Value for check button when not selected

Example:

```
button .hello –text “hello” –command exit
button .goodbye –textvariable goodbye –command exit
set goodbye “So long”
pack .hello .goodbye
```

Example:

```
image create photo img –format gif –file “testing.gif”
button .b –image img
pack .b
```

Example:

```
checkbutton .twoside -text "2 sided" -variable twosided -onvalue 1 -offvalue 0
set twosided 0
```

Example:

```
radiobutton .left -text Left -variable align -value left
radiobutton .center -text Center -variable align -value center
radiobutton .right -text Right -variable align -value right
set align left
```

Example:

```
entry .test -width 6 -border 2 -textvariable testing
pack .test
```

- Frames

A frame is a widget class that provides a container for other widgets.

It's most common use is to break up a larger window into regions; each of which can be arranged with geometry management independently. In turn, the regions can be arranged with geometry management relative to each other.

Example:

```
frame .entry
entry .entry.red -width 7
entry .entry.green -width 7
```

Example:

```
frame .menu -relief raised -borderwidth 2
```

- General Examples

Example:

```
proc PushAction {} {
  puts "Ouch that hurt!";
  exit;
}

button .push -text "Push Me!" -command PushAction;
pack .push;
```

Example:

```
proc power {base p} {
        set result 1

        while {$p > 0} {
                set result [ expr $result * $base ]
                set p [ expr $p - 1 ]
        }

        return $result
}

entry .base -width 6 -relief sunken -textvariable base
label .label1 -text "to the power"
entry .power -width 6 -relief sunken -textvariable power
label .label2 -text "is"
label .result -textvariable result
pack .base .label1 .power .label2 .result -side left -padx 1m -pady 2m
bind .base <Return> { set result [power $base $power]}
bind .power <Return> { set result [power $base $power]}
```

Example (colorset):

```
wm title . "Color Editor"

frame .menu -relief raised -borderwidth 2
menubutton .menu.file -text File -menu .menu.file.m -underline 0
menu .menu.file.m
.menu.file.m add command -label "Exit" -underline 0 -command "destroy ."
pack .menu -side top -fill x
pack .menu.file -side left -fill x

scale .redscale -label Red -from 0 -to 255 -length 7c -orient horizontal -command newColor
scale .greenscale -label Green -from 0 -to 255 -length 7c -orient horizontal -command newColor
scale .bluescale -label Blue -from 0 -to 255 -length 7c -orient horizontal -command newColor
pack .redscale .greenscale .bluescale -side top

frame .entry
entry .entry.red -width 7
entry .entry.green -width 7
entry .entry.blue -width 7
button .entry.set -text Set -command setColor
pack .entry
pack .entry.red .entry.green .entry.blue.entry.set -side left
```

```
canvas .colorwin -width 7c -height 1.5c
pack .colorwin -side bottom -pady 2m

proc newColor {} {
        set color [format #%02x%02x%02x [.redscale get] [.greenscale get] [.bluescale get]]
        .colorwin config -background $color
}

proc setColor {} {
        .redscale set [.entry.red get]
        .greenscale set [.entry.green get]
        .bluescale set [.entry.blue get]
}
```