Introduction to GUI and Event-Driven Programming

01219116 Computer Programming II

Chaiporn Jaikaeo

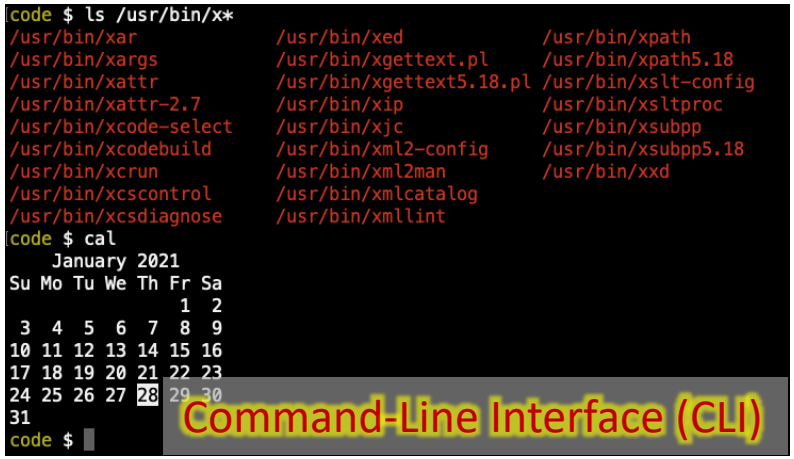
Department of Computer Engineering
Kasetsart University

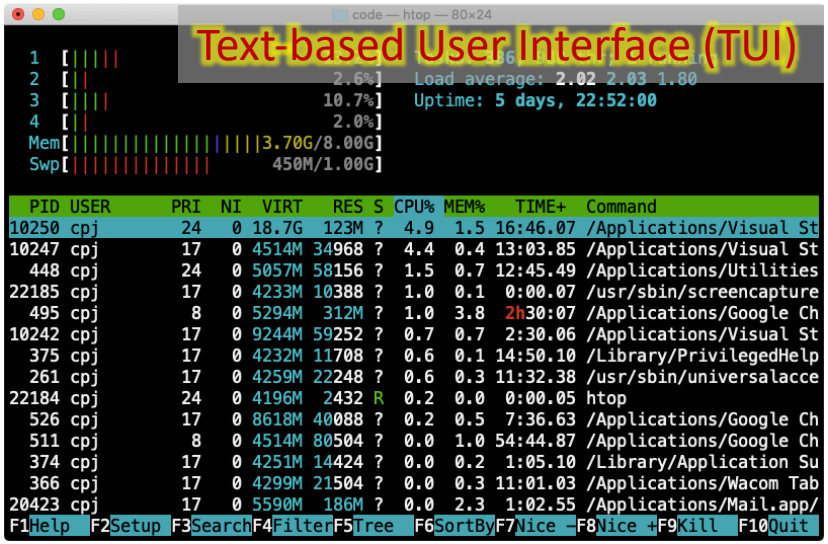
Revised 2021-02-01

Outline

- Python GUI frameworks
- Basics of Tkinter
- GUI widgets
- Widget styling
- Event loop and event-driven programming

GUI vs. CLI vs. TUI





Python GUI Frameworks

- Tkinter
  - Most generic; comes with standard Python
- wxPython
  - Native OS look and feel
  - Platform-specific code may be required
- PyQt/PySide
  - More than GUI framework; supports networking, databases, etc.
  - Platform agnostic
  - Requires separate installation of Qt
- *Many more…*

Tcl/Tk and Tkinter

- Tcl (Tool command language)
  - Designed in 1980s as a general-purpose, interpreted programming language
  - Commonly used for rapid prototyping
- Tk (toolkit) is a cross-platform windowing toolkit for Tcl
- Tkinter is a Python interface to Tcl/Tk

Typical Tkinter Program (No Class)


```python
import tkinter as tk

def say_hello():
    print("Hello")

root = tk.Tk()
root.title("My First GUI App")
root.geometry("300x100")

label = tk.Label(root, text="Welcome to my GUI application")
label.pack()

btn_hello = tk.Button(root, text="Hello", command=say_hello)
btn_hello.pack()

btn_quit = tk.Button(root, text="Quit", command=root.destroy)
btn_quit.pack()

root.mainloop()
```

root window creation

widget creation

running event loop

event handlers (callbacks)

Typical Tkinter Program (with Class)

```python
import tkinter as tk

class App(tk.Tk):

    def __init__(self):
        super().__init__()
        self.title("My First GUI App")
        self.geometry("300x100")
        self.label = tk.Label(self, text="Welcome to my GUI application")
        self.label.pack()
        self.btnHello = tk.Button(self, text="Hello", command=self.say_hello)
        self.btnHello.pack()
        self.btnQuit = tk.Button(self, text="Quit", command=self.destroy)
        self.btnQuit.pack()

    def say_hello(self):
        print("Hello")

    def run(self):
        self.mainloop()

if __name__ == "__main__":
    app = App()
    app.run()
```

The Root Window

- A Tkinter application is initialized by creating one (and only one) root window
- A root window is created by instantiating an object of the Tk class

```
import tkinter as tk

root = tk.Tk()
root.title("My Firt App")
root.geometry("300x200")
```

- For OO style, we can subclass Tk and wrap everything inside our own class

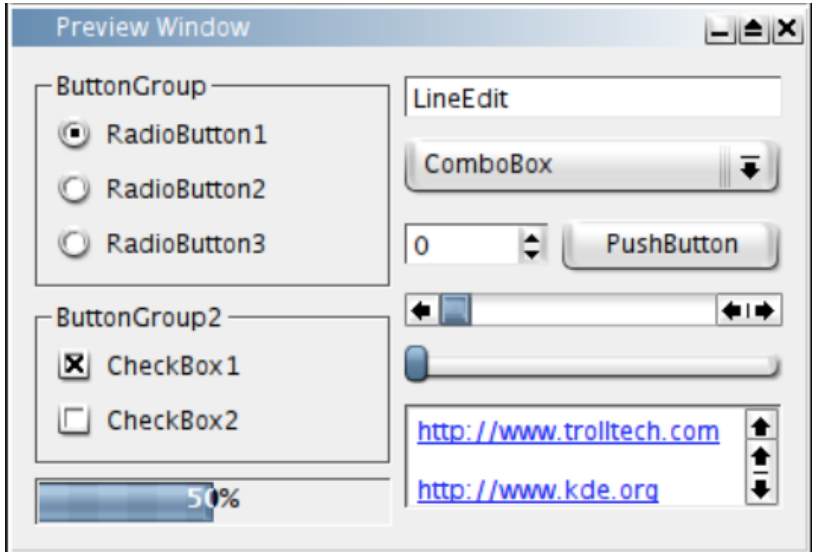
```
import tkinter as tk

class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("My First App")
        self.geometry("300x200")

app = Application()
```

GUI Widgets

- A widget (or a control) is an element of interaction, appearing as a visible part of the application's GUI, such as a button or a scroll bar
- Each widget facilitates a specific type of user-computer interaction



https://en.wikipedia.org/wiki/Graphical_widget

Creating a Widget

- A widget is created by instantiating an object of a widget class
- A list of attributes may also be specified to control the behavior and appearance of the widget
  - See https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/std-attrs.html for more comprehensive list of attributes

```
label1 = tk.Label(text="Name")
label2 = tk.Label(text="School", foreground="blue")
```

- The widget will not show up on screen until one of its layout methods is called
  - we will use the **pack** layout for now

```
label1.pack()
label2.pack()
```

Tk Widgets

Widget	Purpose
Label	Display static text or an image
Button	Execute a specific task; a “do this now” command
Menu	Implement top-level, pulldown, and popup menus
Menubutton	Display popup or pulldown menu items when activated
OptionMenu	Create a popup menu, and a button to display it
Entry	Enter one line of text
Text	Display and edit formatted text, possibly with multiple lines
Checkbutton	Set on-off, True-False selections
Radiobutton	Allow one-of-many selections
Listbox	Choose one or more alternatives from a list
Scale	Select a numerical value by moving a “slider” along a scale
Spinbox	Allow to select from a numerical range or from a list

https://runestone.academy/runestone/books/published/thinkcspy/GUIandEventDrivenProgramming/03_widgets.html

Widget Styling

- A widget can be styled during its creation, or reconfigured later, by setting its appearance-related attributes

```
# style the widget during its creation
lbl = tk.Label(text="Huge Label", font=("Arial",48))

# reconfigure the widget using config() or configure() method
lbl.config(foreground="red")

# widget can also be used as a dict for reconfiguration
lbl["background"] = "yellow"
lbl["text"] = "Fancy Label"
```

Tk Themed Widgets

- The `tkinter.ttk` module provides access to the Tk themed widget set
- Ttk widgets are themed, i.e., their appearances are controlled by a set of configurable styles

```
import tkinter as tk
import tkinter.ttk as ttk

root = tk.Tk()
root.title("Tk vs. Ttk Widgets")
root.geometry("400x100")

btn_tk = tk.Button(text="Tk Button")
btn_tk.pack()

btn_ttk = ttk.Button(text="Ttk Button")
btn_ttk.pack()

root.mainloop()
```

Ttk Widgets

- Ttk implements all original Tk widgets, and provides additional widgets as follows

Widget	Purpose
Combobox	Combine a text field with a pop-down list of values
Progressbar	Show the status of a long-running operation
Separator	Display a horizontal or vertical separator bar
Sizegrip	Allow user to resize the entire application window
Treeview	Display a hierarchical collection of items

Ttk Widget Styling

- Tk widget styling (their styles can be completely independent)

```
label1 = tk.Label(text="Label 1", fg="black", bg="white")
```

- Ttk widget styling (their styles are defined as part of a theme)
  - Define a new style called `BW.TLabel`, which is derived from the default Label style (called `TLabel`)

```
style = ttk.Style()
style.configure("BW.TLabel", foreground="black", background="white")

label1 = ttk.Label(text="Label 1", style="BW.TLabel")
```

  - Make all Label widgets display blue background

```
style.configure("TLabel", background="blue")
```

  - Make every widget use Arial font with size of 24pt

```
style.configure(".", font=("Arial",24))
```

Label Widget

- Display static text or an image
- Typically provide no user interaction

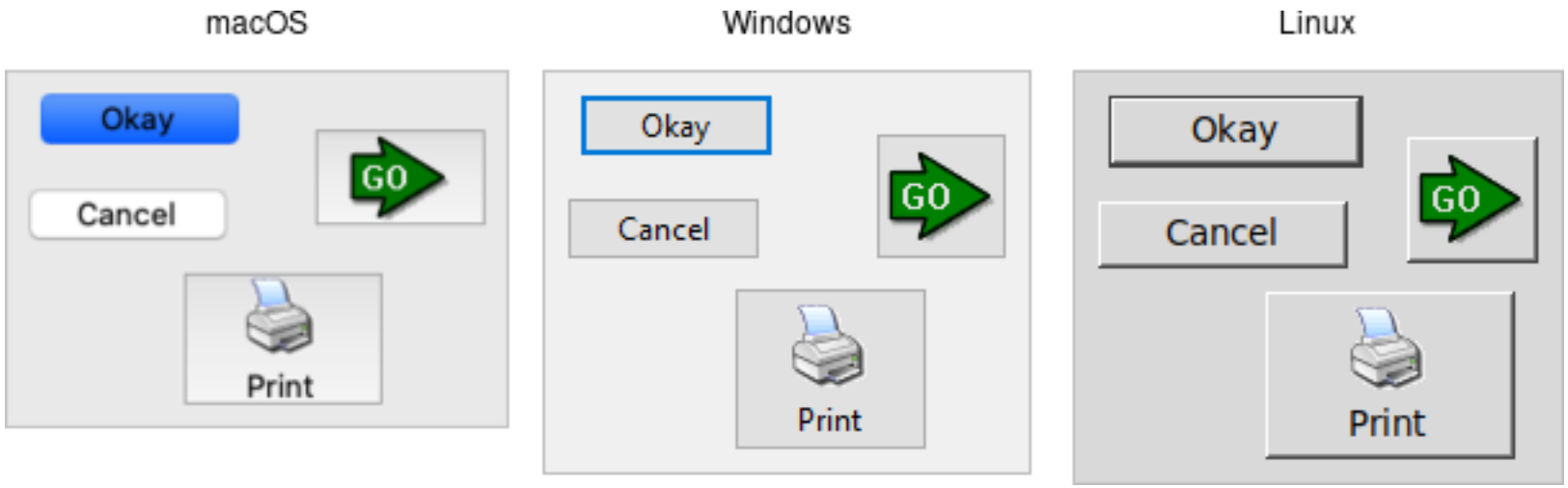
macOS

Windows

Linux

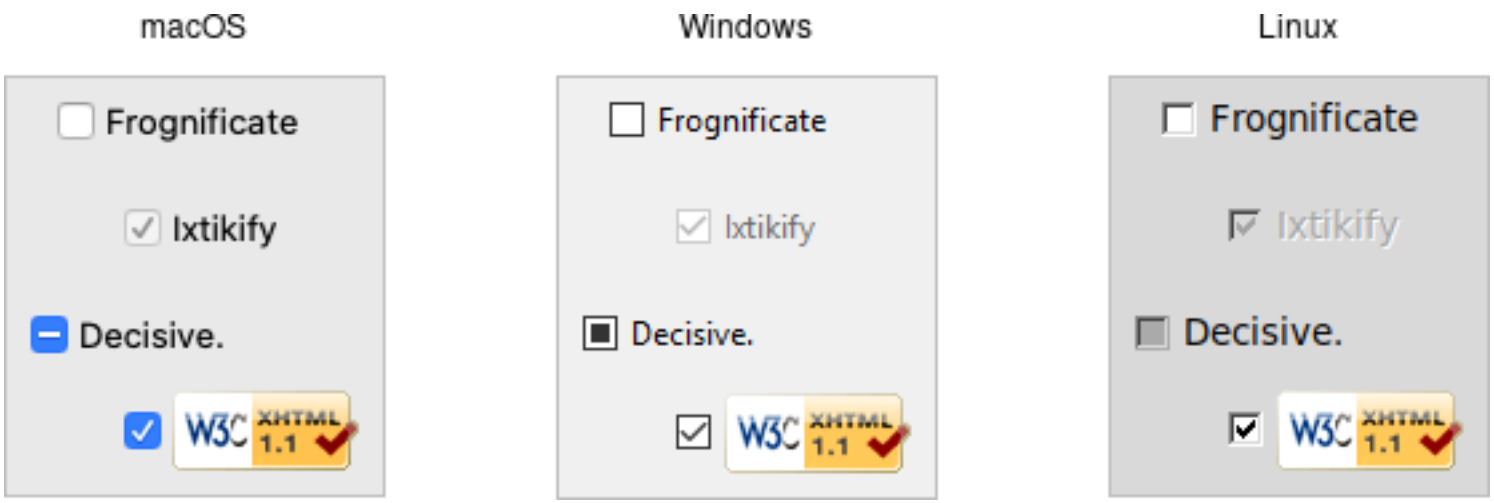
Button Widget

- Execute a specific task; a “do this now” command



Checkbutton Widget

- Set on-off, True-False selections

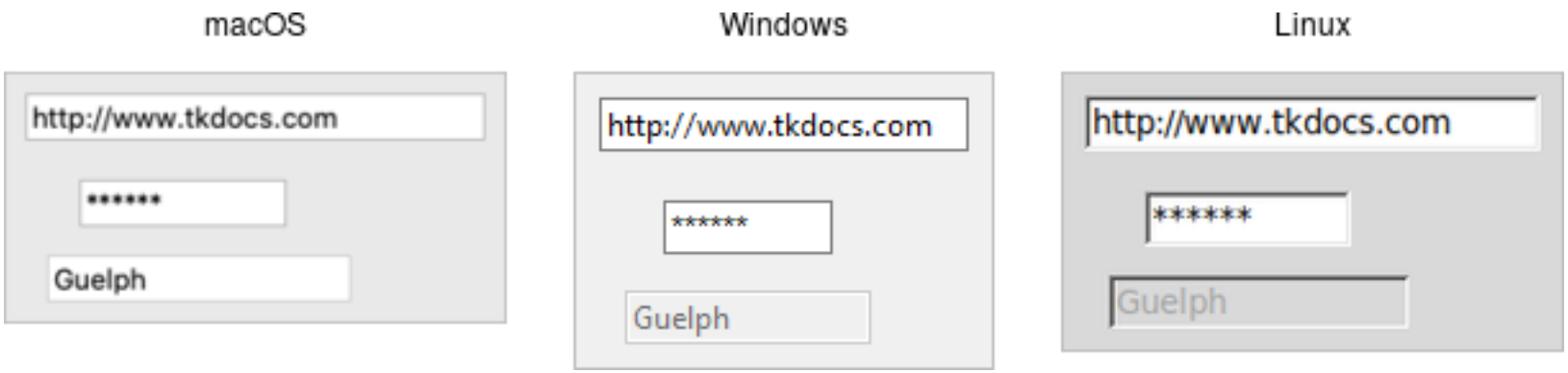


Radiobutton Widget

- Allow one-of-many selections

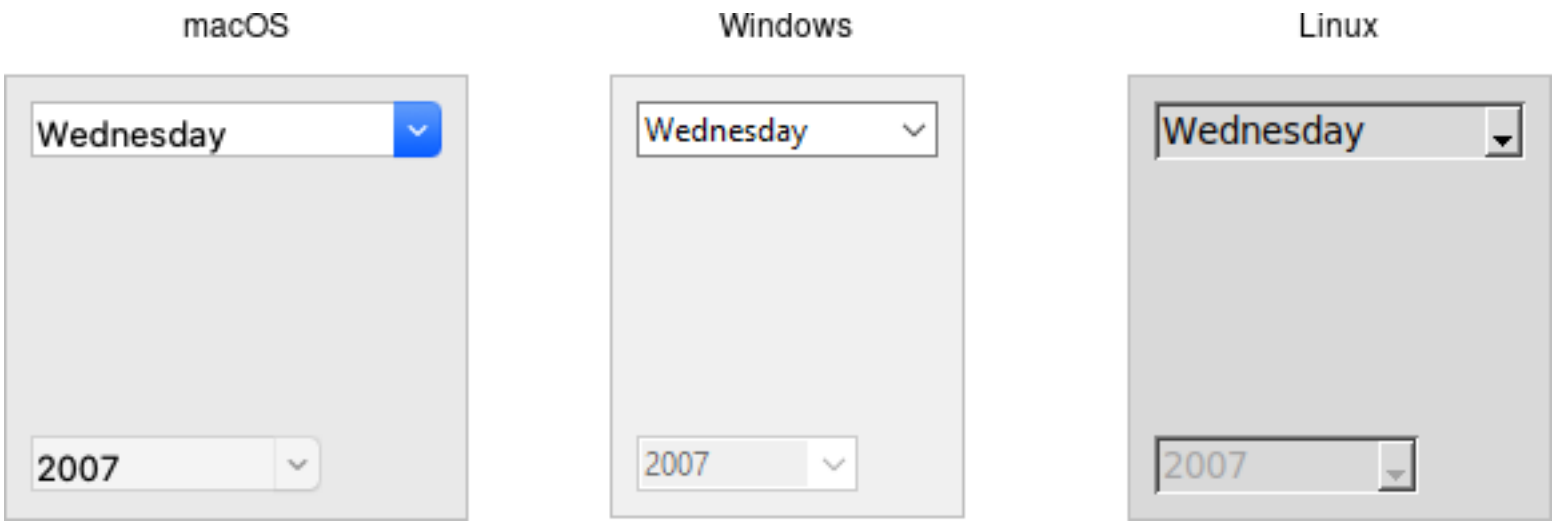
Entry Widget

- Enter one line of text



Combobox Widget

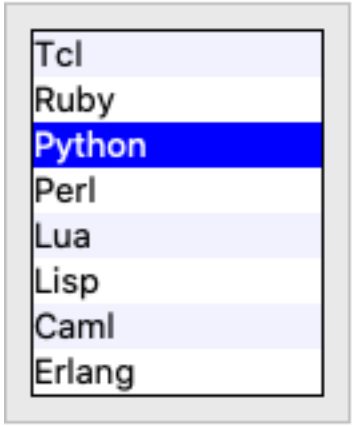
- Combine a text field with a pop-down list of values



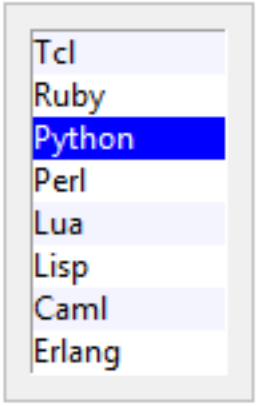
Listbox Widget

- Choose one or more alternatives from a list

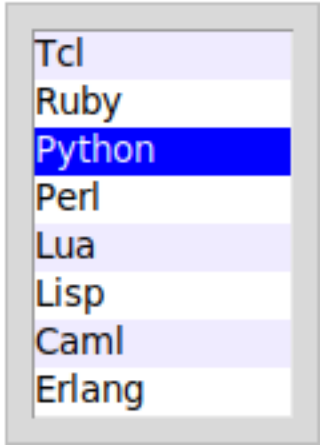
macOS



Windows



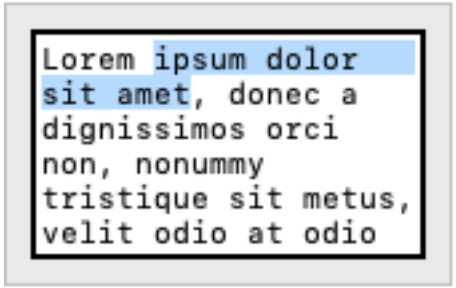
Linux



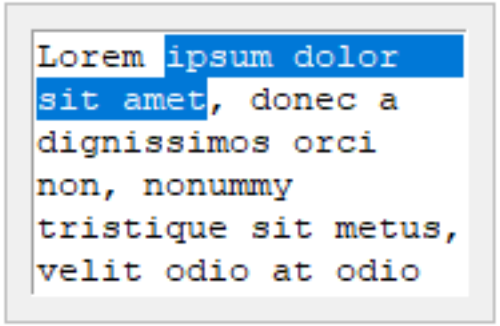
Text Widget

- Display and edit formatted text, possibly with multiple lines

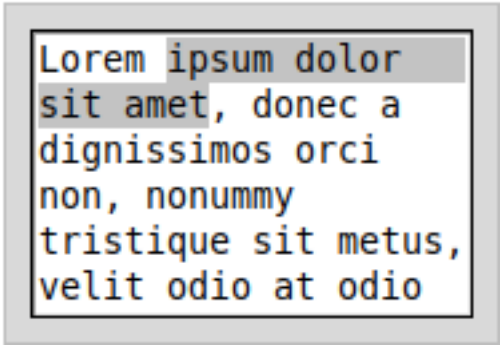
macOS



Windows



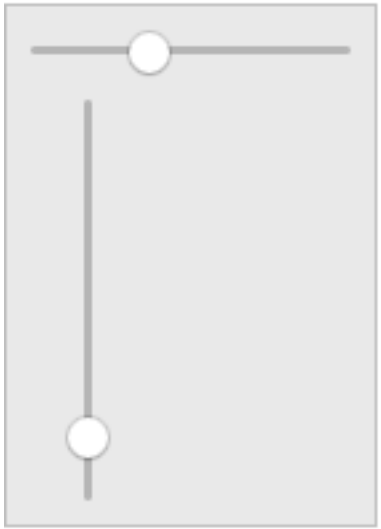
Linux



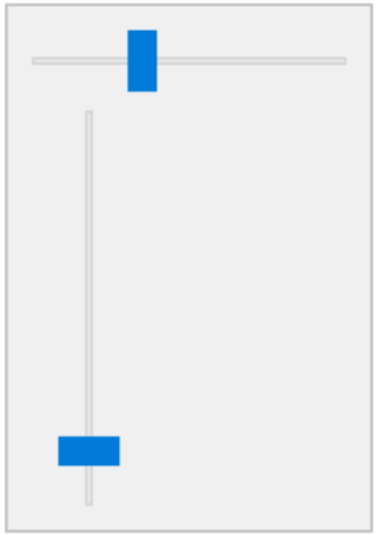
Scale Widget

- Select a numerical value by moving a “slider” along a scale

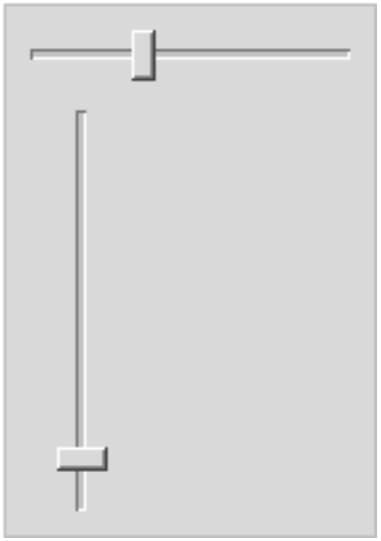
macOS



Windows

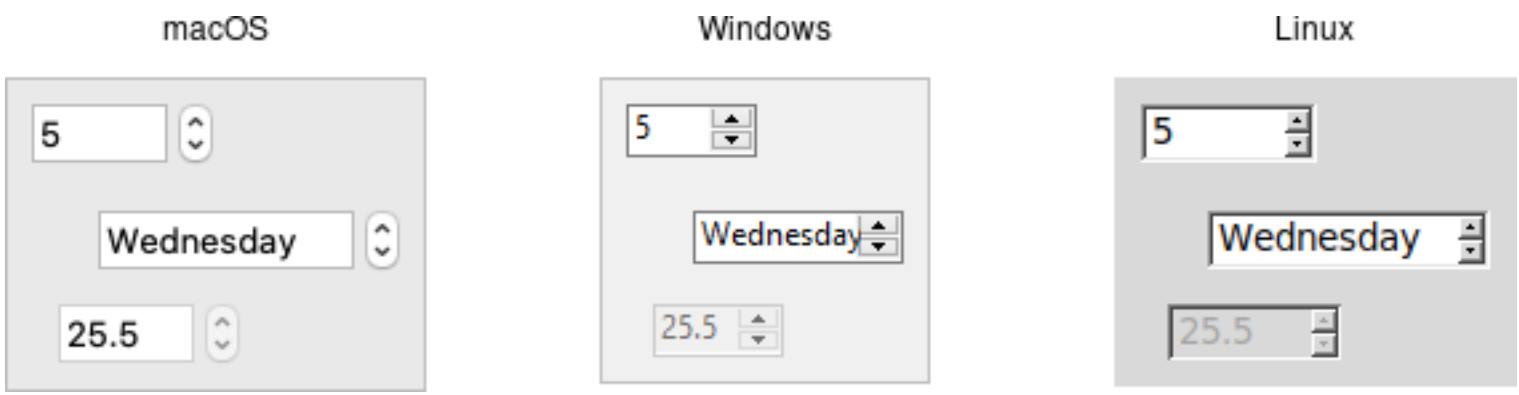


Linux



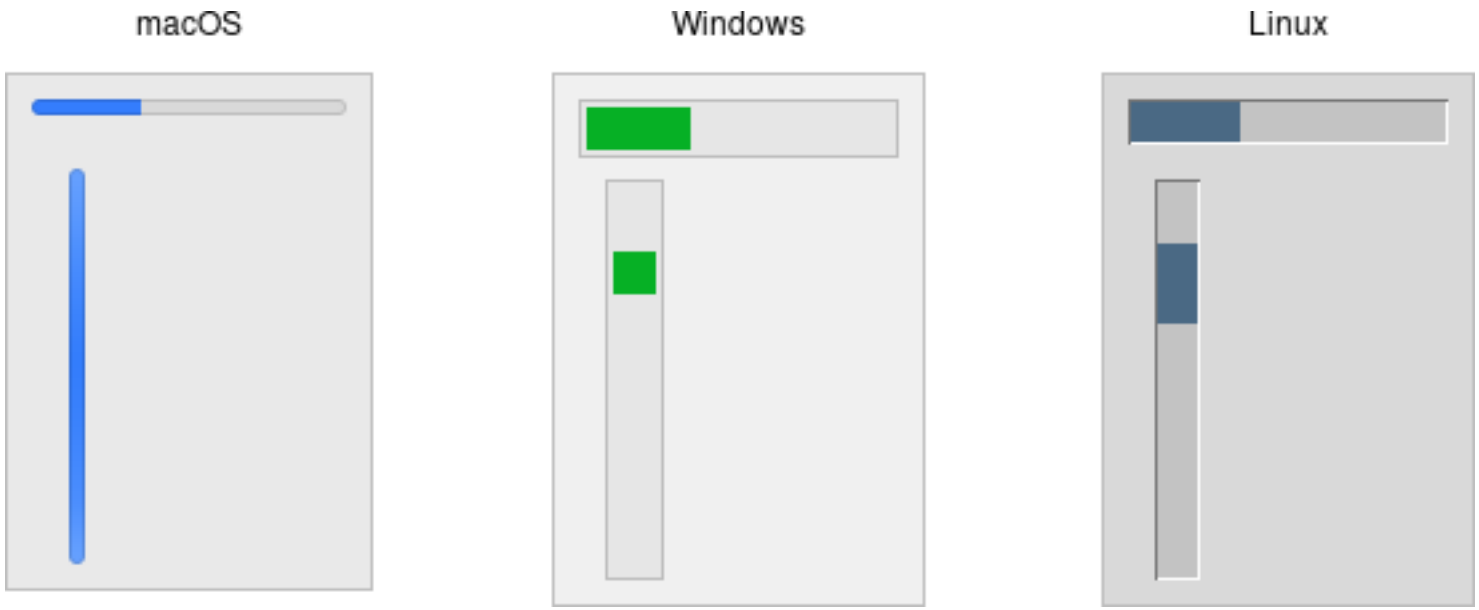
Spinbox Widget

- Allow to select from a numerical range or from a list



Progressbar Widget

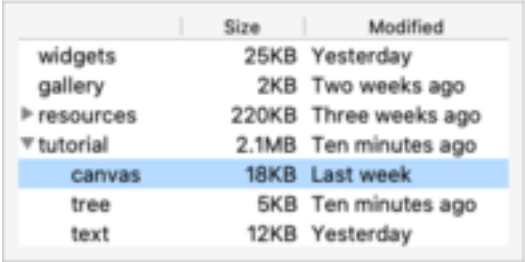
- Show the status of a long-running operation



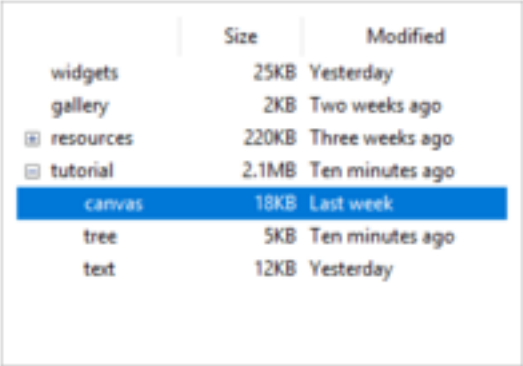
Treeview Widget

- Display a hierarchical collection of items

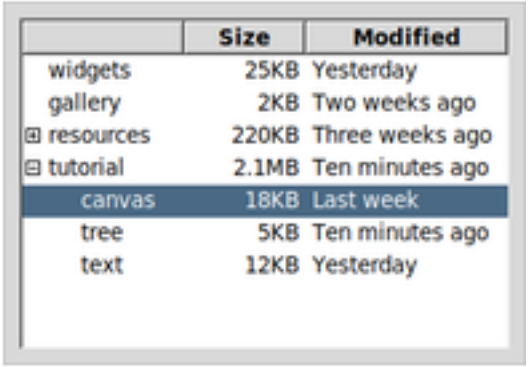
macOS



Windows

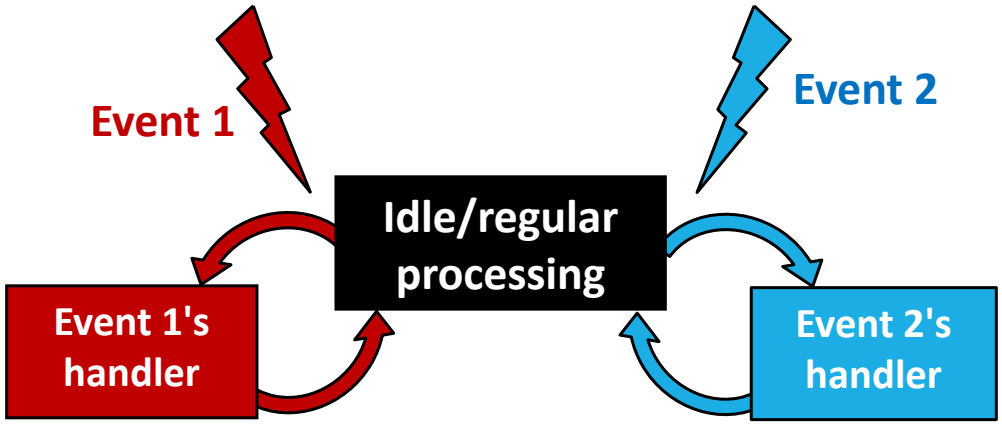


Linux



Event-Driven Programming

- Also known as **reactive programming**
- Perform regular processing or be idle
- React to events when they happen immediately



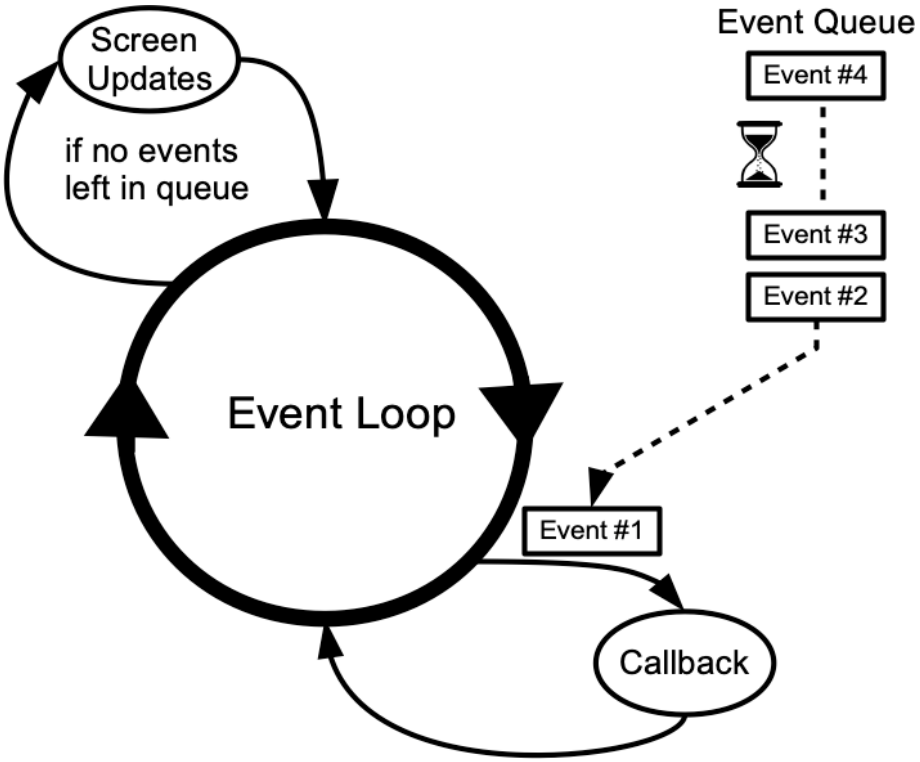
Possible Event Sources

- GUI widgets
- Keyboard and mouse
- Timers
- Control variables

- Most event types can be found from
  - https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/event-types.html

The Event Loop

- The event loop is started by calling the root window's **`mainloop()`** method
  - There can be only one event loop in a Tkinter app



GUI Widget Events

- Certain GUI widgets generate events via user interaction
  - E.g., the Button widget accepts a callback as *command* argument

```
def onclick():
    print("Button is clicked")

btn = ttk.Button(text="Click Me", command=onclick)
```

- Changes of a widget's state such as active, inactive, resized, destroyed, also trigger events

Keyboard and Mouse Events

- Keyboard events are triggered when user presses or releases a key
- Mouse events are triggered when
  - User moves the mouse pointer into, out of, or within a widget
  - User presses or releases a mouse button
- Some keyboard and mouse events' names can be found from:
  - https://www.python-course.eu/tkinter_events_binds.php

Keyboard and Mouse Event Examples

- Run this example and try moving the mouse inside the application or press some keys

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("Event Demonstration")
root.geometry("500x200")

lbl = ttk.Label(text="Starting...", font=("Arial",36))

# make the label fill the container and also expand when the container is resized
lbl.pack(fill=tk.BOTH, expand=1)

lbl.bind("<Leave>", lambda e: lbl.config(text="Moved mouse outside"))
lbl.bind("<Enter>", lambda e: lbl.config(text="Moved mouse inside"))
lbl.bind("<ButtonPress-1>", lambda e: lbl.config(text="Clicked left mouse button"))
lbl.bind("<Double-1>", lambda e: lbl.config(text="Double clicked"))

root.bind("<Return>", lambda e: lbl.config(text="Enter key is pressed"))
root.bind("<Key-F1>", lambda e: lbl.config(text="Key F1 is pressed"))
root.bind("<KeyRelease-F1>", lambda e: lbl.config(text="Key F1 is released"))

root.mainloop()
```

Timers

- All Tk widgets provide the **`after()`** method , whose signature is

```
widget.after(delay, callback=None)
```

- This method calls the function `callback` after the given `delay` in milliseconds
  - If no function is given, it acts like `time.sleep()` (but in milliseconds instead of seconds)
- Example below waits for 2 seconds after the button is clicked, then changes the text to "Done"

```
def onclick():
    btn.config(text="Wait for 2 seconds", state=tk.DISABLED)
    btn.after(2000, lambda: btn.config(text="Done", state=tk.NORMAL))

btn = ttk.Button(text="Start", command=onclick)
```

Control Variable Classes

- Tkinter provides a number of control variable classes that can generate events

Construction	Data Type	Default Value
`tk.StringVar()`	str	`""`
`tk.IntVar()`	int	`0`
`tk.DoubleVar()`	double	`0.0`
`tk.BooleanVar()`	bool	`False`

- The `set()` method must be used to set variable's values
- One may subclass the **`Variable`** class for custom datatypes

Control Variable Event Binding

- Use the **`trace_add()`** method to bind a read/write event to a callback

```
num = tk.IntVar()
num.trace_add("write", lambda *args: print("New value = {}".format(num.get())))
```

- Specific widgets, such as Label and Entry, allow binding to control variables directly
  - A Label widget accepts a StringVar as ***textvariable*** argument and monitors its write events to update the label text
  - An Entry widget accepts a StringVar as ***textvariable*** argument and updates the variable when the text entry changes

```
text = tk.StringVar()
lbl = ttk.Label(textvariable=text)
entry = ttk.Entry(textvariable=text)
```

Conclusion

- GUI provides interactive and friendly interface for users
- Tkinter is a simle GUI toolkit readily available in every Python distribution
- A GUI application consists of one root window containing a number of GUI widgets
- Events may be generated by user interacting with widgets as well as other sources such as variable updates and timers
- An event loop is reponsible for processing events and updating GUI window

References

- *Tkinter* – Python interface to Tcl/Tk
  - https://docs.python.org/3/library/tkinter.html
- Tk themed widgets
  - https://docs.python.org/3/library/tkinter.ttk.html
- Tkinter 8.5 reference: a GUI for Python
  - https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html
- TkDocs Tutorial
  - https://tkdocs.com/tutorial/index.html