Tkinter

CPEN333 – System Software Engineering

2021 W2

University of British Columbia

Introduction

- We will discuss Tkinter which is a Python's standard GUI (Graphic User Interface) toolkit here for three reasons:
  - It is a very good example of object-oriented design and programming.
  - It is a very good example of event-driven programming.
  - And also to serve as a basic introduction to this useful GUI tool which we have also used in the project.
    - Most of the project's GUI code is already provided to you but it is beneficial to understand that code.

Tkinter

- Tkinter is the de-facto standard GUI (Graphic User Interface) toolkit that is traditionally bundled with Python.
  - There are a few Python GUI frameworks, but Tkinter is one of the commonly used ones.
  - Note that there are differences between Tkinter in Python 2, and the one in Python 3. Obviously, we only use Python 3.
- Tkinter is pronounced T-K-Inter (short for tk interface).
  - It is based on tk/tcl which is a GUI library used by many languages for cross-platform GUI development.
- This set of slides serves as a quick intro to the Tkinter essentials.

Why tkinter?

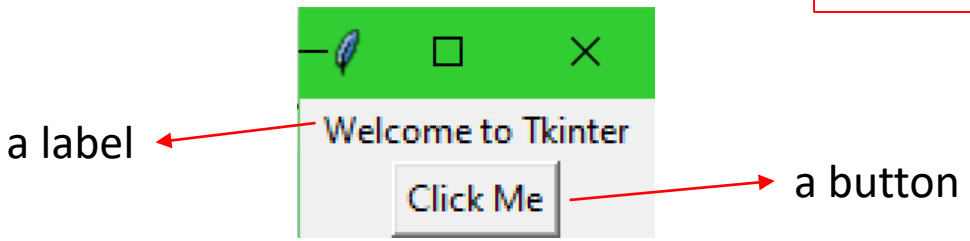
- It is open-source and comes with any Python distribution.
- It is portable: Windows, macOS, Linux
- Learning Tkinter is rather intuitive and one can get the job done quicker.
- Been around for a while → stable and reliable
- *Drawbacks*: depending on the design, it could be slow or not modern-looking.

Getting Started

- Let's start with a simple GUI:

```
from tkinter import * # import all tkinter definitions

window = Tk() # create a root window

# create a label
label = Label(window, text = "Welcome to Tkinter")
label.pack() # place the label in the window

# Create a button
button = Button(window, text = "Click Me")
button.pack() # place the button in the window

window.mainloop() # create an event loop
```

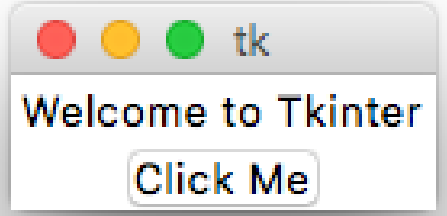
- Which results in:



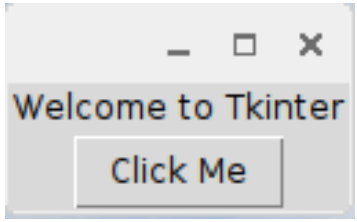
Tkinter's look

- Tkinter will create the window and widgets consistent with the operating system it is running on. This is usually the desired behaviour.

Windows



macOS



Linux

- There is also `ttk` (TK themed widgets): "The basic idea for `tkinter.ttk` is, to the extent possible, to separate the code implementing a widget's behaviour from the code implementing its appearance."

Getting Started (creating a root window)

- The `tkinter` module is included in Python 3 to simply import:

```
from tkinter import *
```

  - The above `import` statement will import all definitions (classes, functions, constants) from tkinter

- We need to create a root window. This can be done by using the Tk class. `Tk()` will create an instance of a window.

```
window = Tk()
```

  - All widgets will be housed in this root window.

- Tkinter provides various controls and visual components that are commonly called widgets.

  - In this example, we have used the `Button` and `Label` widgets.

Getting Started (adding the first widget)

- To display text, we use the `Label` widget.

```
label = Label(window, text = "Welcome to Tkinter")
```

  - The first argument is the parent container, `window`, the root window we created earlier.
  - The `text` property (second argument) is used to specify what text is displayed.

- We can place the label by:

```
label.pack()
```

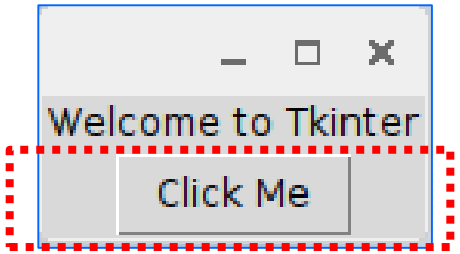
  - `label.pack()` uses the pack manager to manage the placement of the widget in the window.
  - In this example, the widgets are <u>packed row by row</u> in the window.

Getting Started (adding a button)

- Similarly, we use the Button widget to create a simple button.

```
button = Button(window, text = "Click Me")
button.pack()
```

  - The first argument identifies its parent container (again here window) and the `text` property (second argument) is used to set the displayed text for the button.
  - We have used `button.pack()` to pack the button, by placing it on the next row after the label in the window.



  - A button is normally used to execute a command. In the simple example above though, the button has not been bound to any function.
    - So it will do nothing, when it is clicked.

Processing events

- A widget that needs to process an event can be bound to a function, which is called when the event occurs.
  - Such functions are known as callback functions or handlers.
- For example, we can bound a function to a `Button`. This straightforward code demonstrates the basics of event-driven programming.

```
def processCancel():
    print("Cancel button is clicked")

btCancel = Button(window, text = "Cancel", command = processCancel)
btCancel.pack()
```

  - By using `command = processCancel`, we are binding the function `processCancel` to the button.
  - In this simple example, the program will print the message whenever the button is pressed:

```
Cancel button is clicked
Cancel button is clicked
```

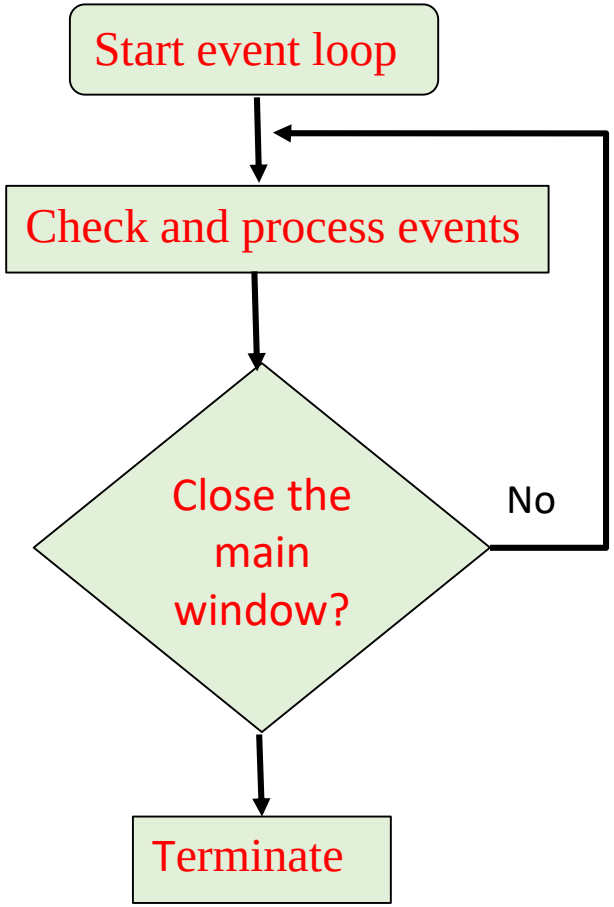
Getting Started (cont.)

- Tkinter is event driven. After the window and widgets are created, the program waits for user interactions.

- We will need to create the event loop. This is done by:

```
window.mainloop()
```

- The event loop repeatedly checks and processes events until the main window is closed.



Tkinter widgets

- Tkinter's GUI classes define common GUI widgets. Each Tkinter class comes with many methods. Examples are:
  - **Button:** A simple button, used to execute a command.
  - **Label:** Displays text or an image.
  - **Entry:** A text entry field, also called a text field or a text box.
  - **Frame** A container widget for containing other widgets.
  - **Canvas:** Structured graphics, used to draw graphs and plots, create graphics editors, and implement custom widgets.
  - **Checkbutton**: clicking a check button toggles between the values
  - **Menu**: A menu pane, use to implement pull-down and popup menus.
  - **Menubutton**: A menu button, used to implement ull-down menus.
  - **Message**: Displays text. Similar to label but can automatically wrap text
  - **Radiobutton**: Clicking a radio button sets the variable to that value and clears all other radio buttons associated with the same variable.
  - **Text**: Formatted text display. Allows to display and edit text with various styles.
  - … (https://coderslegacy.com/python/python-gui/)

Button and Label widgets

- We have already used these two widgets.

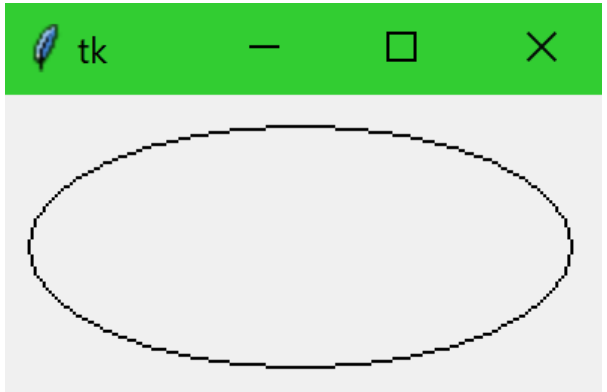
- Button:
  - https://coderslegacy.com/python/python-gui/python-tkinter-button/

- Label:
  - https://coderslegacy.com/python/python-gui/python-tkinter-label/

Tkinter Canvas Widget

- We can use the `Canvas` widget to display shapes.
- The following example creates a `canvas` of 200x100 pixels.

```
from tkinter import *
window = Tk()
canvas = Canvas(window, width = 200, height = 100)
canvas.pack()
canvas.create_oval(10, 10, 190, 90)
window.mainloop()
```

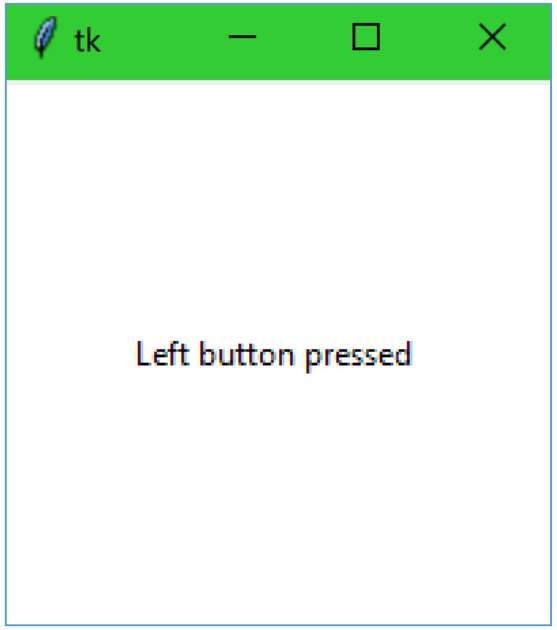


- Tkinter Canvas has many methods to create different shapes such as:
  - `create_line`: to create a line
  - `create_rectangle`: to create a rectangle
  - `create_oval`, `create_arc`, `create_text`, `create_polygon`, …
  - `delete`: to delete a shape
- https://coderslegacy.com/python/tkinter-canvas/

Binding mouse buttons

- We can use the method `widget.bind(event, handler)` to bind an event to a callback handler.
- <Button-1> is the left, and <Button-3> is the right mouse button.

```
from tkinter import *

def left(event):
   clear()
   canvas.create_text(100, 100, text="Left button pressed", tags="left")
def right(event):
   clear()
   canvas.create_text(100, 100, text="Right button pressed", tags="right")
def clear():
   canvas.delete("left", "right")

window = Tk()
canvas = Canvas(window, width = 200, height = 200, bg = "white")
canvas.pack()
canvas.bind("<Button-1>", left)
canvas.bind("<Button-3>", right)

window.mainloop()
```



config() method

- We can use the `config` method to modify a widget's option/property. For example, assuming

```
from tkinter import *
root = Tk()

valueLabel= Label(root, text ="")
valueLabel.pack()
```

  - Then we can change the displayed text of the label to display the value stored in the variable `i` by using the `config` method as follows:

```
i = 0
valueLabel.config(text = i)
```

- An alternative is to use

```
valueLabel["text"] = i
```

Tkinter Geometry Managers

- A *geometry manager* is used to place widgets inside a container.

- Tkinter has three geometry managers:
  - Pack manager
  - Grid manager
  - Place manager

- Here we only focus on the pack manager, for brevity.

Pack manager

- This is the simplest one and we have used it so far many times.
- It can place widgets on top of each other, or side by side (using the `side` option of the `pack()` method).

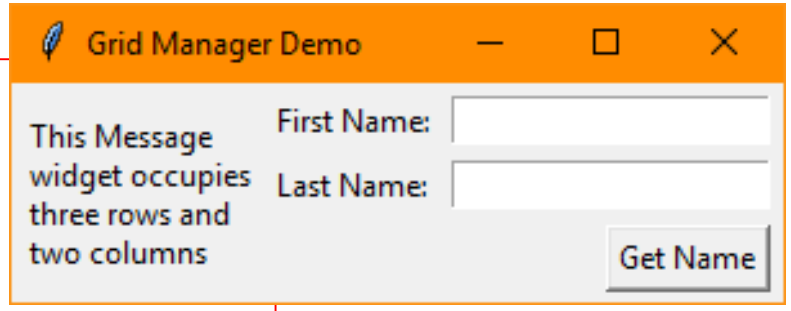
```
from tkinter import *
root = Tk()

bt = Button(root, text="OK").pack()
valueLabel= Label(root, text ="Hello world!").pack()
bt2 = Button(root, text="Cancel").pack()

root.mainloop()
```

- https://www.tutorialspoint.com/python/tk_pack.htm

Using Classes

- We can use classes for OOP in Tkinter:

```
from tkinter import * # import tkinter

class GridManagerDemo:
    window = Tk() # create a window
    window.title("Grid Manager Demo") # set title

    message = Message(window, text =
        "This Message widget occupies three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)
    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)
    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)
    Button(window, text = "Get Name").grid(row = 3, padx = 5,
          pady = 5, column = 4, sticky = E) # ToDo: add callback

    window.mainloop() # Create an event loop

GridManagerDemo() # Create GUI
```



Should be improved by defining methods

The *after* method

- When we are using `tkinter`, we are using its event loop.
- We can use `tkinter`'s `after` method to schedule/register a callback function that is called after a given time in milliseconds.
  - For example, here the `after` method is used to schedule a callback to the function every 2000 *ms*.

```
after(delay_ms, callback_func)
```

```
from tkinter import *
root = Tk()

#some code here

def doSomething ():
    #some code here
    root.after(2000, doSomething)

#rest of the code here
```

References

- D. Liang's ***Introduction to Programming using Python***
- Tkinter:
  - https://docs.python.org/3/library/tkinter.html
  - https://wiki.python.org/moin/TkInter
    - https://coderslegacy.com/python/python-gui/
  - https://www.w3schools.in/python-tutorial/gui-programming/
  - https://www.python-course.eu/tkinter_labels.php
  - https://www.tutorialspoint.com/python3/python_gui_programming.htm
- Python GUI Programming:
  - https://wiki.python.org/moin/GuiProgramming