Python DTS PRoA 2022

Library Session 1: Tkinker

Muhammad Ogin Hasanuddin

KK Teknik Komputer
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Pendahuluan

- GUI programming in Python.
- Tkinter is pronounced as tea-kay-inter. Tkinter is the Python interface to Tk, which is the GUI toolkit for Tcl/Tk.
- Tcl (pronounced as tickle) is a scripting language often used in testing, prototyping, and GUI development. Tk is an open-source, cross-platform widget toolkit used by many different programming languages to build GUI programs.
- Python implements the Tkinter as a module. Tkinter is a wrapper of C extensions that use Tcl/Tk libraries.
- Tkinter allows you to develop desktop applications. It’s a very good tool for GUI programming in Python.
- Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI.

Why Tkinter

- Easy to learn.
- Use very little code to make a functional desktop application.
- Layered design.
- Portable across all operating systems including Windows, macOS, and Linux.
- Pre-installed with the standard Python library.

Outline

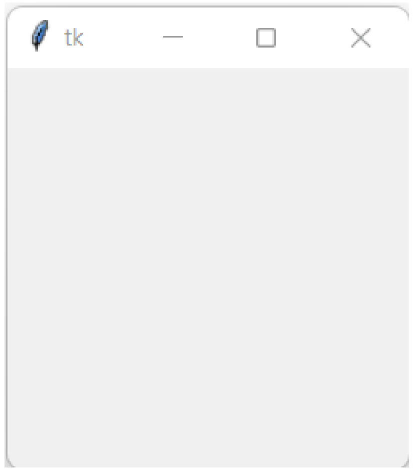
- Tkinter fundamental
- Tk themed widgets
- Example: Temperature Converter application

Creating a window

- Berikut code yang digunakan untuk menampilkan window ke layer

```
import tkinter as tk

root = tk.Tk()
root.mainloop()
```

- Jika program dieksekusi akan muncul tampilan seperti berikut



How it works?

- First, import the tkinter module as tk to the program

```
import tkinter as tk
```

- Second, create an instance of the tk.Tk class that will create the application window:

```
root = tk.Tk()
```

- By convention, the main window in Tkinter is called root. But you can use any other name like main
- Third, call the mainloop() method of the main window object:

```
root.mainloop()
```

mainloop()

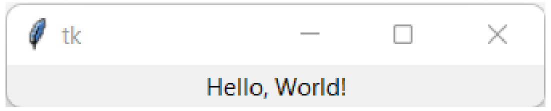
- The mainloop() keeps the window visible on the screen. If you don't call the mainloop() method, the window will display and disappear immediately. It will be so fast that you may not see its appearance.
- Also, the mainloop() method keeps the window displaying and running until you close it.
- Typically, you place the call to the mainloop() method as the last statement in a Tkinter program, after creating the widgets.

Displaying a label

- Now, it's time to place a component on the window. In Tkinter, components are called widgets.
- The following adds a label widget to the root window:

```
import tkinter as tk


root = tk.Tk()

# place a label on the root window
message = tk.Label(root, text="Hello, World!")
message.pack()

# keep the window displaying
root.mainloop()
```

- If you run the program, you'll see the following output:



```
Hello, World!
```

How it works?

- To create a widget that belongs to a container, you use the following syntax:

```
widget = WidgetName(container, **options)
```

- In this syntax:
  - The `container` is the parent window or frame that you want to place the widget.
  - The `options` is one or more keyword arguments that specify the configurations of the widget.
- In the program, the following creates a Label widget placed on the root window:

```
message = tk.Label(root, text="Hello, World!")
```

- And the following statement positions the Label on the main window:

```
message.pack()
```

- If you don't call the `pack()` method, the Tkinter still creates the widget. However, the widget is invisible.

Fixing the blur UI on Windows

- If you find the text and UI are blurry on Windows, you can use the ctypes Python library to fix it.
- First import the ctypes module:

```
from ctypes import windll
```

- Second, call the SetProcessDpiAwareness() function:

```
windll.shcore.SetProcessDpiAwareness(1)
```

- If you want the application to run across platforms such as Windows, macOS, and Linux, you can place the above code in a try...finally block:

```
try:
    from ctypes import windll

    windll.shcore.SetProcessDpiAwareness(1)
finally:
    root.mainloop()
```

Summary

- Import tkinter module to create a Tkinter desktop application.
- Use Tk class to create the main window and call the mainloop() method to keep the window displays.
- In Tkinter, components are called widgets.

Tkinter Window

- As we seen before, the root window has a title that defaults to tk. It also has three system buttons including Minimize, Maximize, and Close.
- In this part, we will learn how to manipulate various attributes of a Tkinter window.

Changing the window title

- To change the window's title, you use the title() method like this:

```
window.title(new_title)
```

- For example, the following changes the title of the root window to 'Tkinter Window Demo':

```
import tkinter as tk

root = tk.Tk()
root.title('Tkinter Window Demo')

root.mainloop()
```

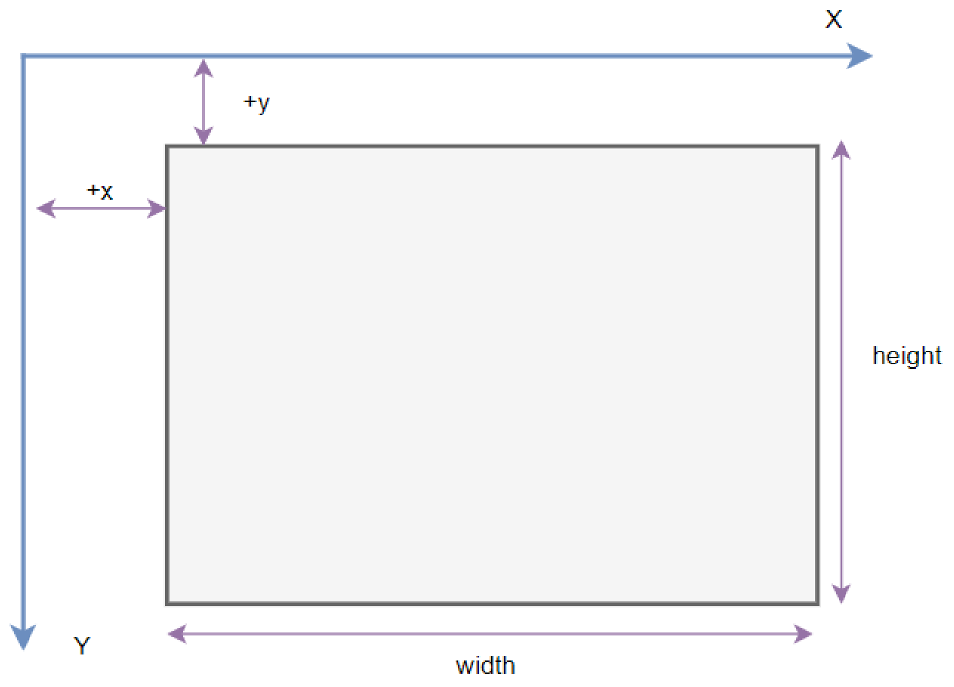
- To get the current title of a window, you use the title() method with no argument:

```
title = window.title()
```

Window size and location

- In Tkinter, the position and size of a window on the screen is determined by geometry. The following shows the geometry specification:

$$width \times height \pm x \pm y$$



```
window.geometry('widthxheight±x±y')
```

Window size and location

- The width is the window’s width in pixels.
- The height is the window’s height in pixels.
- The x is the window’s horizontal position. For example, +50 means the left edge of the window should be 50 pixels from the left edge of the screen. And -50 means the right edge of the window should be 50 pixels from the right edge of the screen.
- The y is the window’s vertical position. For example, +50 means the top edge of the window should be 50 pixels below the top of the screen. And -50 means the bottom edge of the window should be 50 pixels above the bottom of the screen.

Changing window size and location

- To change the size and position of a window, you use the geometry() method:

```
window.geometry(new_geometry)
```

- The following example changes the size of the window to 600x400 and the position of the window to 50 pixels from the top and left of the screen:

```
import tkinter as tk


root = tk.Tk()
root.title('Tkinter Window Demo')
root.geometry('600x400+50+50')

root.mainloop()
```

Changing window size and location

- Sometimes, you may want to center the window on the screen. The following program illustrates how to do it:

```
import tkinter as tk


root = tk.Tk()
root.title('Tkinter Window - Center')

window_width = 300
window_height = 200

# get the screen dimension
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

# find the center point
center_x = int(screen_width/2 - window_width / 2)
center_y = int(screen_height/2 - window_height / 2)

# set the position of the window to the center of the screen
root.geometry(f'{window_width}x{window_height}+{center_x}+{cente
r_y}')


root.mainloop()
```

How it works?

- First, get the screen width and height using the `winfo_screenwidth()` and `winfo_screenheight()` methods.
- Second, calculate the center coordinate based on the screen and window width and height.
- Finally, set the geometry for the root window using the `geometry()` method.

- If you want to get the current geometry of a window, you can use the `geometry()` method without providing any argument:

```
window.geometry()
```

Resizing behavior

- By default, you can resize the width and height of a window. To prevent the window from resizing, you can use the resizable() method:

```
window.resizable(width,height)
```

- The resizable() method has two parameters that specify whether the width and height of the window can be resizable.
- The following shows how to make the window with a fixed size:

```
import tkinter as tk

root = tk.Tk()
root.title('Tkinter Window Demo')
root.geometry('600x400+50+50')
root.resizable(False, False)

root.mainloop()
```

Resizing behavior

- When a window is resizable, you can specify the minimum and maximum sizes using the minsize() and maxsize() methods:

```
window.minsize(min_width, min_height)
window.maxsize(min_height, max_height)
```

Transparency

- Tkinter allows you to specify the transparency of a window by setting its alpha channel ranging from 0.0 (fully transparent) to 1.0 (fully opaque):

```
window.attributes('-alpha',0.5)
```

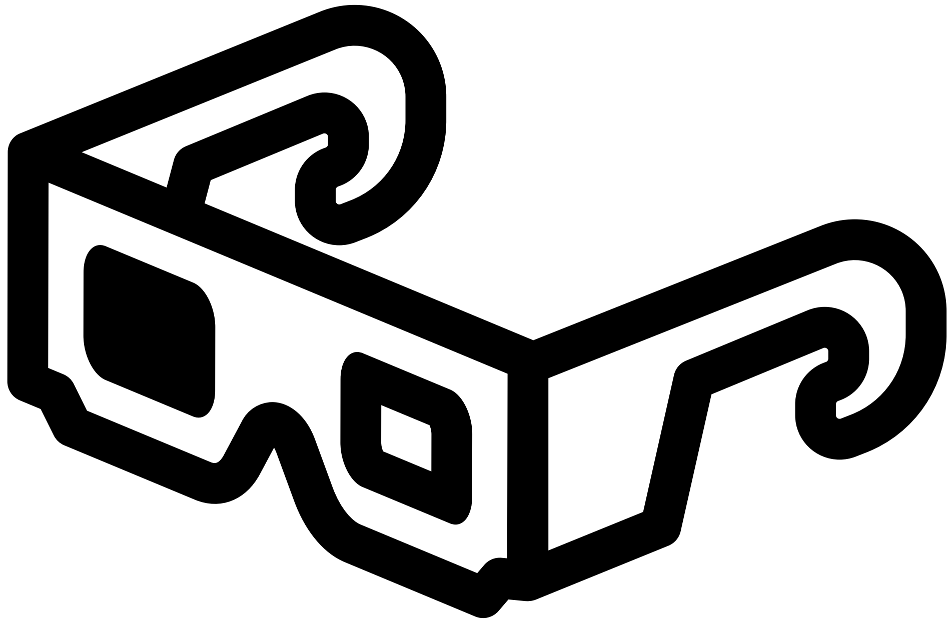
- The following example illustrates a window with 50% transparent:

```
import tkinter as tk

root = tk.Tk()
root.title('Tkinter Window Demo')
root.geometry('600x400+50+50')
root.resizable(False, False)
root.attributes('-alpha', 0.5)

root.mainloop()
```

Changing the default icon

- Tkinter window displays a default icon. To change this default icon, you follow these steps:

1. Prepare an image in the .ico format. If you have the image in other formats like png or jpg, you can convert it to the .ico format. There are many online tools that allow you to do it quite easily.
2. Place the icon in a folder that can be accessible from the program.
3. Call the iconbitmap() method of the window object.



Changing the default icon

- The following program illustrates how to change the default icon to a new one:

```python
import tkinter as tk


root = tk.Tk()
root.title('Tkinter Window Demo')
root.geometry('300x200+50+50')
root.resizable(False, False)
root.iconbitmap('new_icon.ico')

root.mainloop()
```

Summary

- Use the title() method to change the title of the window.
- Use the geometry() method to change the size and location of the window.
- Use the resizable() method to specify whether a window can be resizable horizontally or vertically.
- Use the window.attributes('-alpha',0.5) to set the transparency for the window.
- Use the window.attributes('-topmost', 1) to make the window always on top.
- Use lift() and lower() methods to move the window up and down of the window stacking order.
- Use the iconbitmap() method to change the default icon of the window.

Tk themed widgets

- Tkinter has two generations of widgets:
  - The old classic tk widgets. Tkinter introduced them in 1991.
  - The newer themed ttk widgets added in 2007 with Tk 8.5. The newer Tk themed widgets replace many (but not all) classic widgets.
- Note that ttk stands for Tk themed. Therefore, Tk themed widgets are the same as ttk widgets
- The tkinter.ttk module contains all the new ttk widgets. It's a good practice to always use themed widgets whenever they're available.
- The following statements import the classic and the new Tk themed widgets:

```
import tkinter as tk         # classic
from tkinter import ttk      # new Tk themed
```

Create classic and themed labels

- Example code for create classic and themed labels

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()

tk.Label(root, text='Classic Label').pack()
ttk.Label(root, text='Themed Label').pack()

root.mainloop()
```

Tkinter button widget

- Button widgets represent a clickable item in the applications. Typically, you use a text or an image to display the action that will be performed when clicked.
- Buttons can display text in a single font. However, the text can span multiple lines. On top of that, you can make one of the characters underline to mark a keyboard shortcut.
- To invoke a function or a method of a class automatically when the button is clicked, you assign its command option to the function or method. This is called the command binding in Tkinter.
- To create a button, you use the ttk.Button constructor as follows:

```
button = ttk.Button(container, **option)
```

- A button has many options. However, the typical ones are like this:

```
button = ttk.Button(container, text, command)
```

  - The `container` is the parent component on which you place the button.
  - The `text` is the label of the button.
  - The `command` specifies a callback function that will be called automatically when the button clicked.

Command callback

- The command option associates the button's action with a function or a method of a class. When you click or press the button, it'll automatically invoke a callback function.
- To assign a callback to the command option, you can use a lambda expression:

```
def callback():
    # do something


ttk.Button(
   root,
   text="Demo Button",
   command=callback
)
```

- If the function contains one expression, you use a lamba expression:

```
ttk.Button(
   root,
   text="Demo Button",
   command=lambda_expression
)
```

Button states

- The default state of a button is normal. In the normal state, the button will respond to the mouse events and keyboard presses by invoking the callback function assigned to its command option.
- The button can also have the disabled state. In the disabled state, a button is greyed out and doesn’t respond to the mouse events and keyboard presses.
- To control the state of a button, you use the state() method:

```
# set the disabled flag
button.state(['disabled'])

# remove the disabled flag
button.state(['!disabled'])
```

Simple Tkinter button example

- The following program shows how to display an Exit button. When you click it, the program is terminated.

```
import tkinter as tk
from tkinter import ttk

# root window
root = tk.Tk()
root.geometry('300x200')
root.resizable(False, False)
root.title('Button Demo')

# exit button
exit_button = ttk.Button(
    root,
    text='Exit',
    command=root.quit
)

exit_button.pack(
    ipadx=5,
    ipady=5,
    expand=True
)

root.mainloop()
```

Tkinter image button example

- The following program shows how to create download button using image

```
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo

# root window
root = tk.Tk()
root.geometry('300x200')
root.resizable(False, False)
root.title('Image Button Demo')

# download button
def download_clicked():
    showinfo(title='Information’, message='Download button clicked!')

download_icon = tk.PhotoImage(file='./assets/download.png')
download_button = ttk.Button(root, image=download_icon, command=download_clicked)

download_button.pack(ipadx=5, ipady=5, expand=True)

root.mainloop()
```

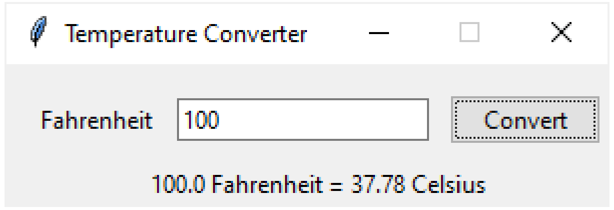
Displaying an image button

```python
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo

# root window
root = tk.Tk()
root.geometry('300x200')
root.resizable(False, False)
root.title('Image Button Demo')

# download button handler
def download_clicked():
    showinfo(title='Information', message='Download button clicked!')

download_icon = tk.PhotoImage(file='./assets/download.png')

download_button =
ttk.Button(root,image=download_icon,text='Download',compound=tk.LEFT,command=download_clicked)

download_button.pack(ipadx=5, ipady=5, expand=True)

root.mainloop()
```

Summary

- Use the ttk.Button() class to create a button.
- Assign a lambda expression or a function to the command option to respond to the button click event.
- Assign the tk.PhotoImage() to the image property to display an image on the button.
- Use the compound option if you want to display both text and image on a button.

Temperature Converter application



100.0 Fahrenheit = 37.78 Celsius

- Basically, the application has a label, an entry, and a button. When you enter a temperature in Fahrenheit and click the Convert button, it'll convert the value in the textbox from Fahrenheit to Celsius.
- If you enter a value that cannot be converted to a number, the program will show an error.
- First, import the tkinter module, ttk submodule, and the showerror function from tkinter.messagebox:

```
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showerror
```

- Second, create the root window and set its configurations:

```
# root window
root = tk.Tk()
root.title('Temperature Converter')
root.geometry('300x70')
root.resizable(False, False)
```

Temperature Converter application

- Third, define a function that converts a temperature from Fahrenheit to Celsius:

```
def fahrenheit_to_celsius(f):
    """ Convert fahrenheit to celsius
    """
    return (f - 32) * 5/9
```

- Fourth, create a frame that holds form fields:

```
frame = ttk.Frame(root)
```

- Fifth, define an option that will be used by all the form fields:

```
options = {'padx': 5, 'pady': 5}
```

- Sixth, define label:

```
temperature_label = ttk.Label(frame, text='Fahrenheit')
temperature_label.grid(column=0, row=0, sticky='W', **options)
```

Temperature Converter application

- Seventh, define the entry, and button. The label will show the result once you click the Convert button:

```
# temperature entry
temperature = tk.StringVar()
temperature_entry = ttk.Entry(frame, textvariable=temperature)
temperature_entry.grid(column=1, row=0, **options)
temperature_entry.focus()

# convert button

def convert_button_clicked():
    """  Handle convert button click event
    """
    try:
        f = float(temperature.get())
        c = fahrenheit_to_celsius(f)
        result = f'{f} Fahrenheit = {c:.2f} Celsius'
        result_label.config(text=result)
    except ValueError as error:
        showerror(title='Error', message=error)

convert_button = ttk.Button(frame, text='Convert')
convert_button.grid(column=2, row=0, sticky='W', **options)
convert_button.configure(command=convert_button_clicked)
```

Temperature Converter application

- result label

```
# result label
result_label = ttk.Label(frame)
result_label.grid(row=1, columnspan=3, **options)
```

- Finally, place the frame on the root window and run the mainloop() method:

```
# add padding to the frame and show it
frame.grid(padx=10, pady=10)

# start the app
root.mainloop()
```

Terimakasih!