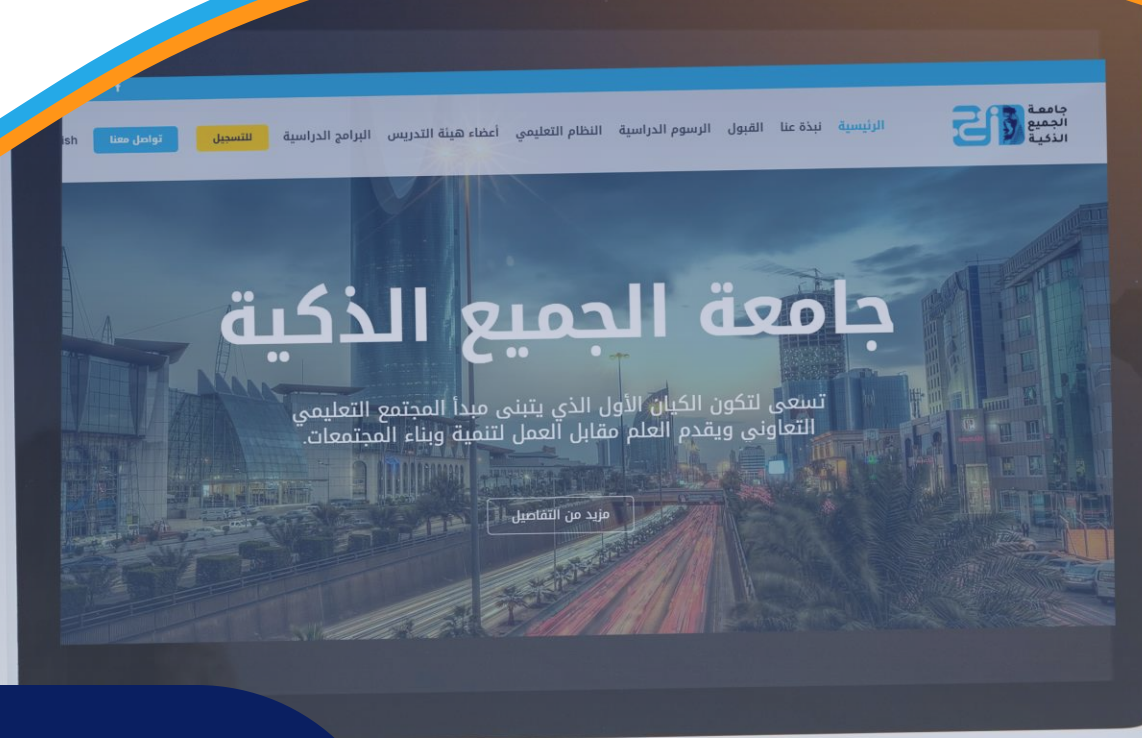




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

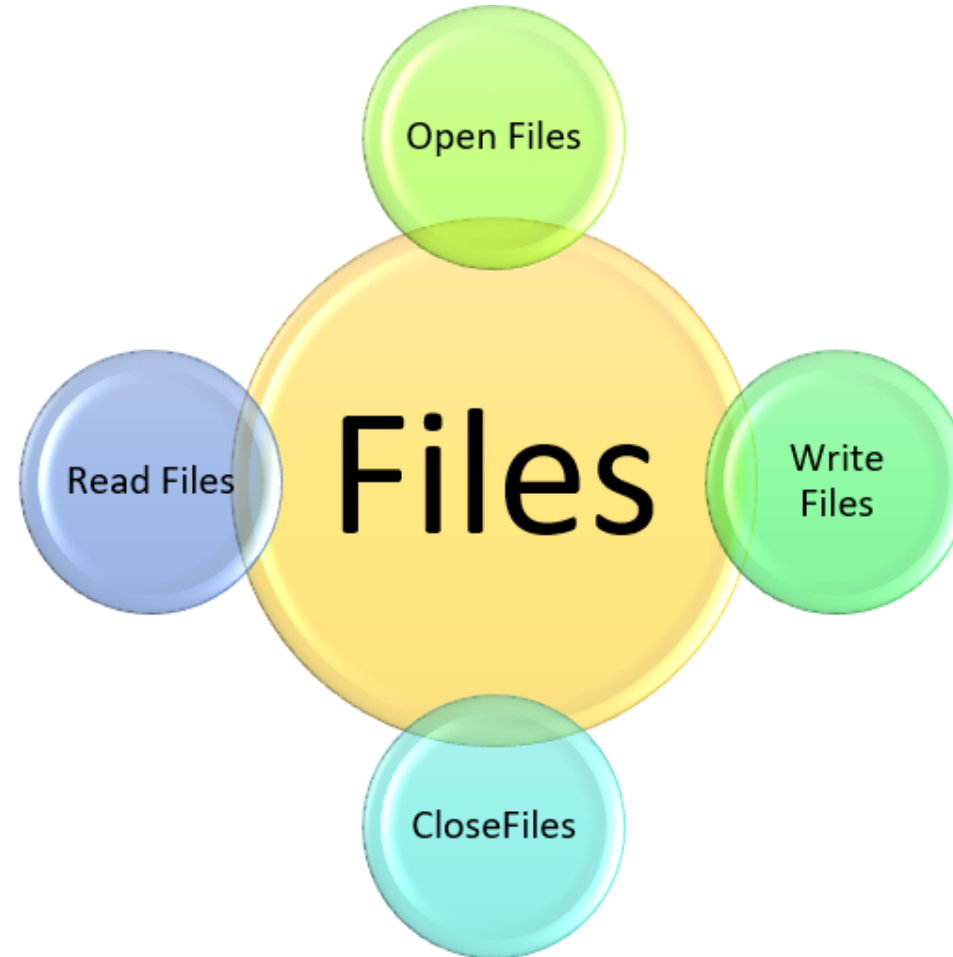


File Handling and GUI
Prepared by Dr. Hamada I. AbdulWakel
Lecture 10

CYB550
Programming Foundations In
Python

- Explain the meaning of file handling.
- Learn the file operations.
- Know the difference between Terminal-based and graphical user interface programs.
- Learn the Tkinter GUI framework.

- **Files** are named locations on disk to store related information.
- They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since RAM is volatile (which loses its data when the computer is turned off).



- The syntax of opening a file is:
`File_object_name = open(filename, mode)`
- The "filename" is a string argument which specify filename along with its path.
- The mode is a string argument which used to specify how file will be used.

- Modes of opening a file:
 - "r" Open a file for reading.
 - "w" Open a file for writing.
 - "a" Open a file for appending.

- When we are done with performing operations on the file, we need to properly close the file.
- Closing a file will **free up the resources** that were tied with the file.
- The syntax for closing a file is:

`File_object_name.close()`

- 
- If an exception occurs when we are performing some operation with the file, the code exits without closing the file. So, we use `try .. finally` statement.

`try:`

`File_object_name = open(filename, mode)`

`finally:`

`File_object_name.close()`

- The best way to close a file is by using the `with` statement. This ensures that the file is closed when the block inside the `with` statement is exited.

`with open(filename, mode) as f:`
 #file operations

- We must open the file in reading **r** mode.
- To read data back from the file you need one of these methods:
 - **read([number])** : return specified number of characters from the file. If the number is omitted, it will read the entire content.
 - **readline()** : returns the next line of the file.
 - **readlines()** : read all the lines as a list of strings.

- Example:

```
import os
os.chdir("/media/dell/Work/python codes/Lec5Codes")
f = open("myfile.txt", "r")
print(f.read(10))
print(f.readline())
print(f.read())
f.close()
```

- Example:

```
lines = f.readlines()  
for i in lines:  
    print(i)
```

- We need to open it in write **w**, append **a** or exclusive creation **x** mode.
- We need to be careful with the w mode, as it will overwrite into the file if it already exists.

- Example:

```
import os
```

```
os.chdir("/media/dell/Work/python codes/Lec5Codes")
```

```
with open("myfile.txt", "w") as f:
```

```
    f.write("My Name: Hamada I.")
```

```
    f.write("I'm a teacher at FCI")
```

- Example:

```
import os
```

```
os.chdir("/media/dell/Work/python codes/Lec5Codes")
```

```
with open("myfile.txt", "a") as f:
```

```
    f.write("I have 3 wonderful kids")
```

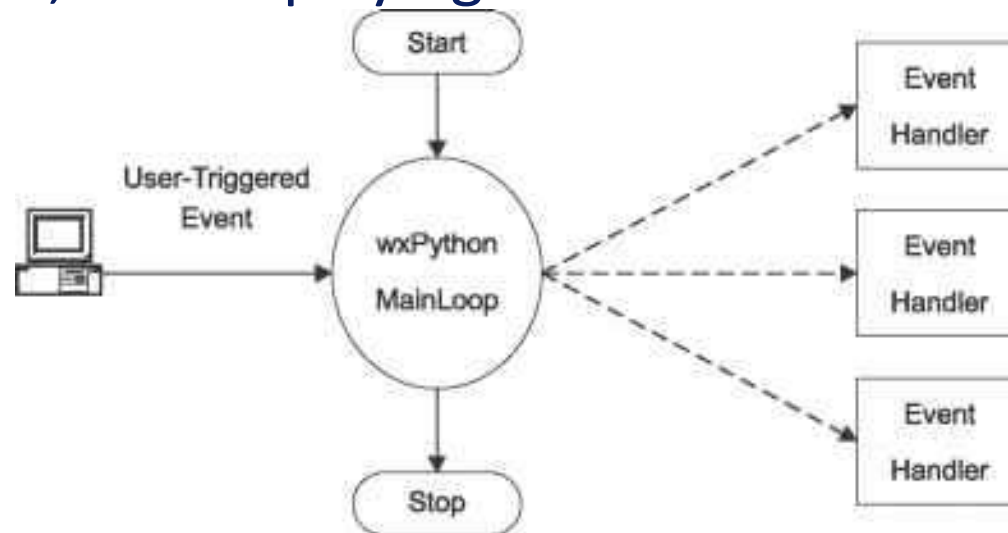


- Program versions:
 - The terminal-based version.
 - The GUI version.
- Both programs perform exactly the same function.
- Their behavior, or look and feel, from a user's perspective are quite different



- GUI-based program:
 - It is user-friendly and simple.
 - It is attractive for non-technical users.
 - No need to execute commands for carrying out functions like opening file , saving etc.
 - Look and feel is better than text interface
 - User can switch quickly between tasks on the GUI interface.

- In GUI, The user-generated **events** (i.e., mouse clicks, button clicks) to **trigger** operations in the program to respond by pulling in inputs, processing them, and displaying results.





- For GUI, there are a number of frameworks used to create UI:
 - PyGUI
 - Wax
 - Pyforms
 - PySimpleGUI
 - Tkinter
 -, and more

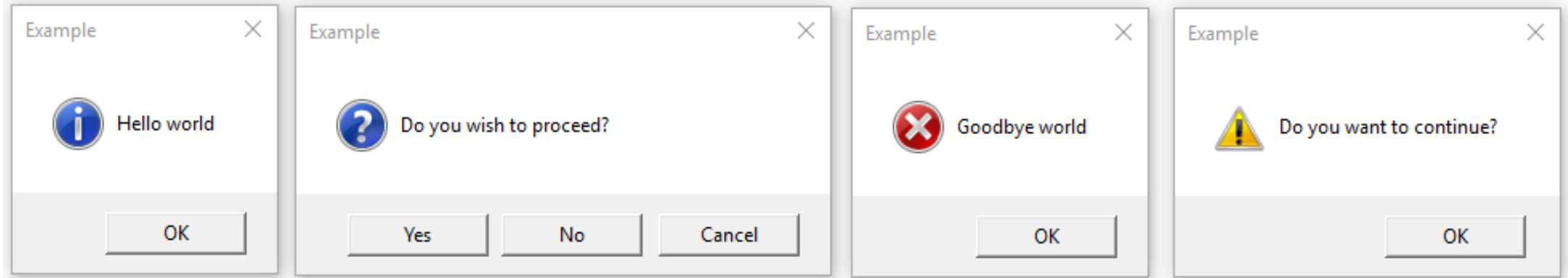


- Tkinter is included with Python as a library. To use it:
`from tkinter import *`
- Tkinter gives you the ability to create Windows with widgets in them.
- A widget is a graphical component on the screen.



- Containers: include frame, labelframe, and paned window.
- Buttons: include buttons, radio buttons, check buttons, and menu buttons.
- Text: include labels and messages.
- Entry: include scale, scrollbar, Listbox, and optionmenu.

- You can create alert boxes using the messagebox method.
- The messagebox module is used to display the message boxes.





- Example:

```
from tkinter import messagebox
```

```
messagebox.showinfo("title", "Information")
```

```
messagebox.showwarning("title", "Warning")
```

```
messagebox.showerror("title", "Error")
```

```
messagebox.askquestion("title", "Do you love GUI?")
```

```
response = messagebox.askyesnocancel("title", "Do you love GUI?")
```

```
print(response)
```

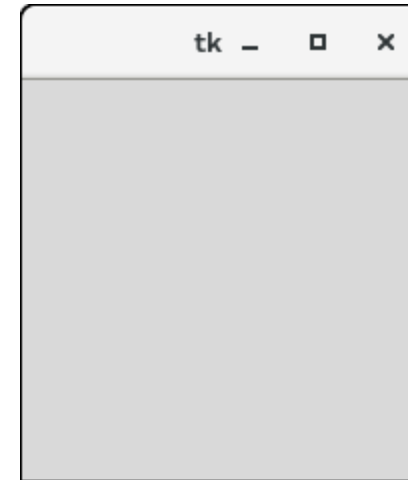


- 
- A frame is basically a rectangular area that can contain other widgets.
 - An empty Tkinter top-level window can be created by using the following steps:
 - import the Tkinter module.
 - Create the main application window.
 - Add the widgets like labels, buttons, frames, etc. to the window.
 - Call the main event loop so that the actions can take place on the user's computer screen.



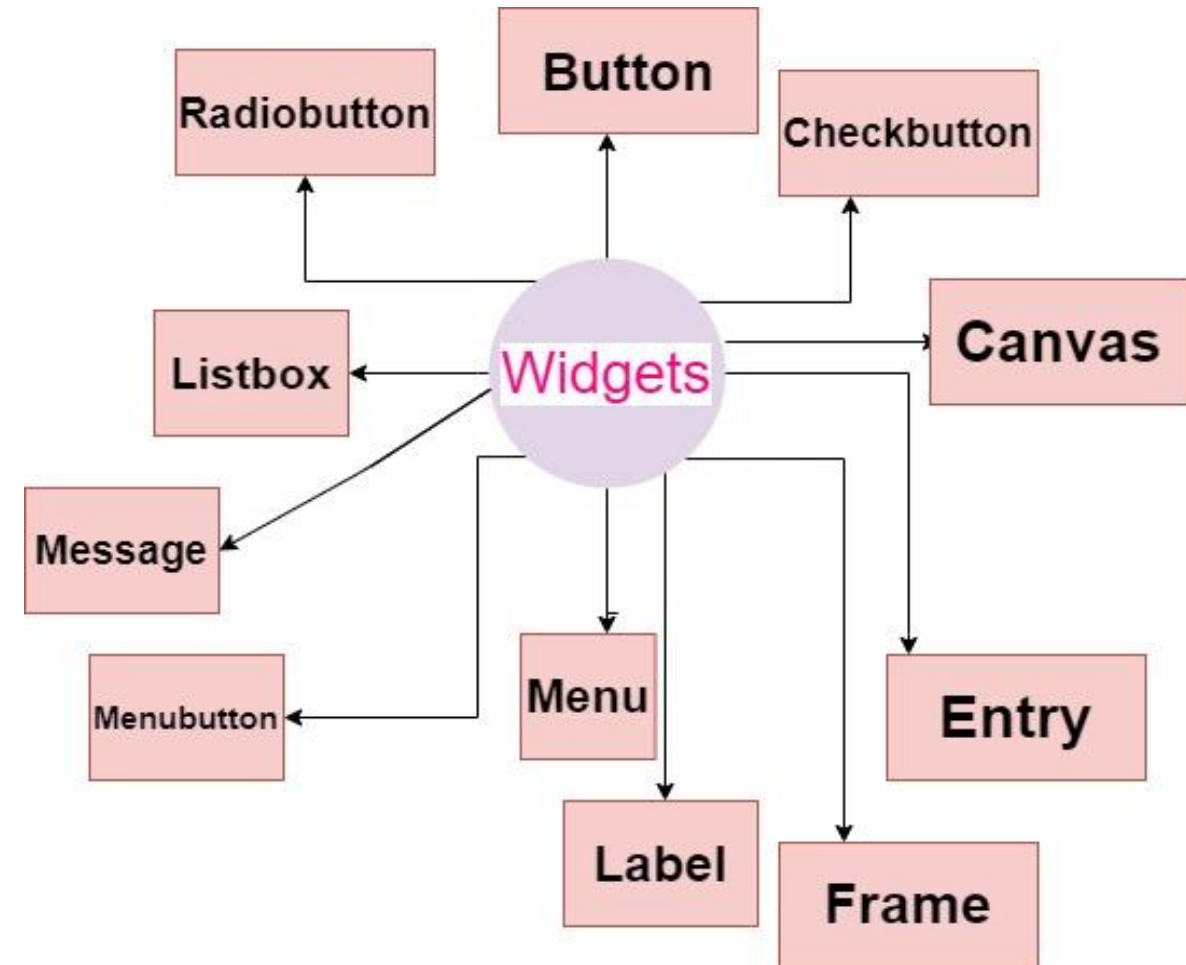
- Example:

```
from tkinter import *  
  
mainFrame = Tk()  
mainFrame.title("The main frame title")  
mainFrame.geometry("300x300")  
mainFrame.mainloop() #listens for events
```



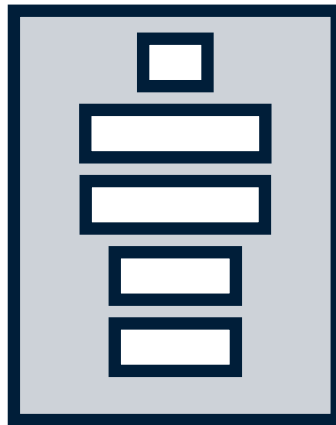
Lecture 10

Widgets



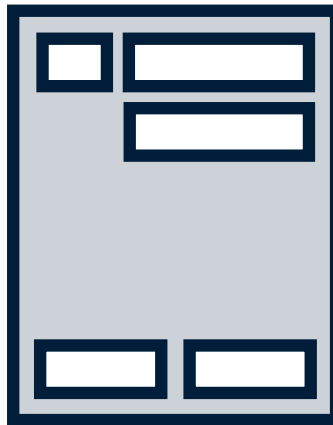


- The geometry measurements allow you to organize the widgets and throughout the parent frames or parent widget area.



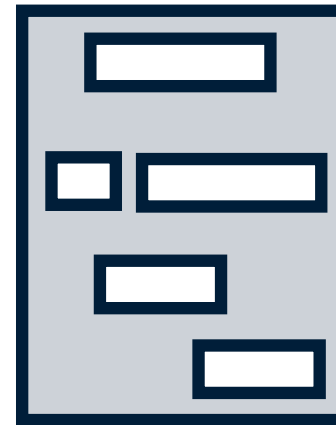
Pack

One after another,
vertically or horizontally



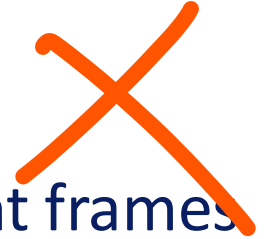
Grid

Assigned to grid cells,
using rows and columns



Place

Put stuff anywhere,
using coordinates



- The geometry allows you to organize the widgets on the parent frames.
 - **pack()**: It organizes the widgets in a block manner, and the complete available width is occupied by it.
`widget.pack(options)`
 - **grid()**: It organizes the widgets in a table-like structure.
`widget.grid(options)`
 - **place()**: Its purpose is to place the widgets at a specific position as instructed by the user in the parent widget.
`widget.place(options)`



- The grid layout uses the matrix row-column concept.

00	01
10	11

- Grid primarily takes two parameters row and column. Imagine 00 corresponds to the first widget.



- Example:

```
username = Label(root, text="User Name:").pack(side=TOP)
```

```
username = Label(root, text="User Name:").grid(row=0)
```

```
username = Label(root, text="User Name:").place(x = 30,y = 50)
```



- Any widget can be added using this syntax:

```
widget_var_name = widgetName(parent, option = placeholder)
```

Where **parent** is the argument for the parent frame while **option** is a placeholder that can have various values differ from widget to the other

- Example:

```
btnOk = Button(root, text="Ok", fg="red", bd=2, bg="light blue")  
usName = Entry(root, width=20).pack()
```

```
var1 = IntVar()  
var2 = IntVar()
```

```
ChkBttn1 = Checkbutton(root, text="Option 1", variable=var1, onvalue=1, offvalue=0).pack()  
ChkBttn2 = Checkbutton(root, text="Option 2", variable=var2, onvalue=1, offvalue=0).pack()
```





- Example:

```
var1 = StringVar()
```

```
RBttn = Radiobutton(root, text = "Option1", variable = var1, value = 1).pack()
```

```
RBttn2 = Radiobutton(root, text = "Option2", variable = var1, value = 2).pack()
```

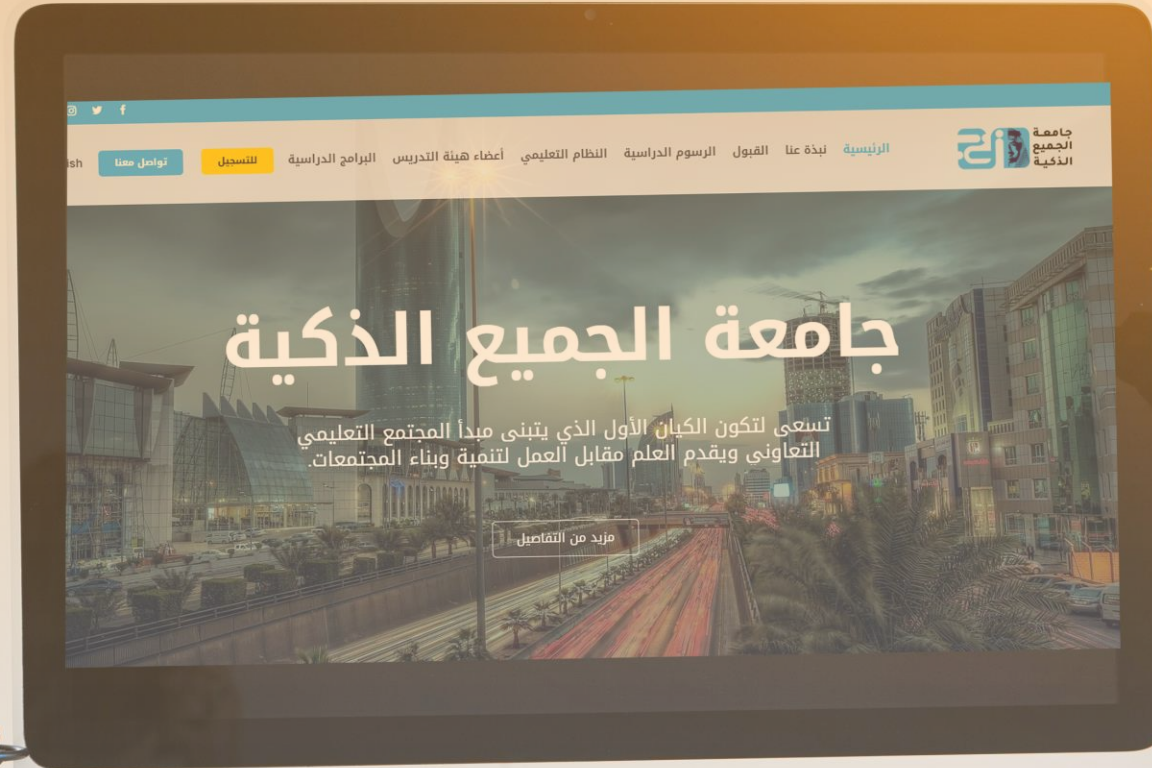
```
label = Label(root, text = "Label text" ).pack()
```

```
from tkinter import ttk
```

```
vlist = ["Option1", "Option2", "Option3", "Option4", "Option5"]
```

```
Combo = ttk.Combobox(root, values = vlist).pack()
```





جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THANK YOU