Lecture 10 - Python Graphical User Interface(GUI)

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

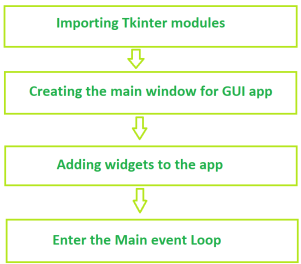
Introduction to Tkinter

Tkinter is the inbuilt Python module that is used to create GUI applications. It is one of the most commonly used modules for creating GUI applications in Python.

- Simple and easy to work with.
- No installation.
- Object-oriented interface.

Introduction to Tkinter

Fundamental structure of tkinter program:



Widgets in Tkinter are the elements of the GUI application that provide various controls to users to interact with the application.

Introduction to Tkinter

An example project of tkinter:

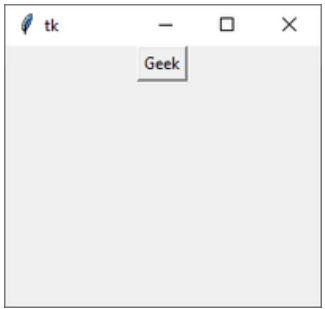
```
from tkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
button = Button(frame, text ='Geek')
button.pack()
root.mainloop()
```

1. Create root window.
2. Create frame inside root window and call geometry method.
3. Create button inside frame which is inside root.
4. Call Tkinter event loop.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Introduction to Tkinter

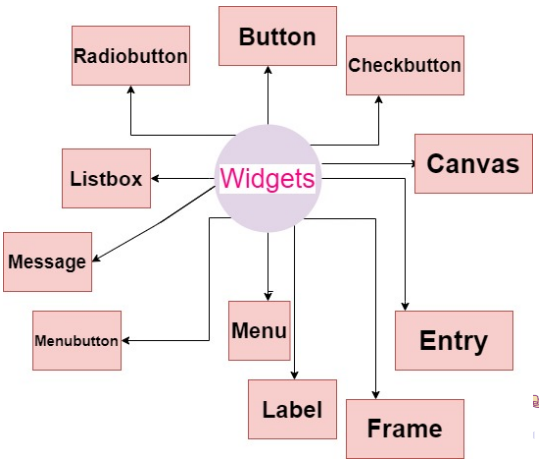
An example project of tkinter:



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Widgets in Tkinter

Widgets in Tkinter are the elements of the GUI application that provide various controls to users to interact with the application. The core widget classes are:



Widgets in Tkinter

Geometry Management. Creating a new widget doesn't mean that it will appear on the screen. To display it, we need to call a special method:

- pack(): The Pack geometry manager packs widgets in rows or columns.
- grid(): The Grid geometry manager puts the widgets in a 2-dimensional table.
- place(): The Place geometry manager allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window.

Widgets in Tkinter

Feature	.pack()	.grid()	.place()
Description	Automatically arranges widgets in a sequence (top, bottom, left, right).	Arranges widgets in a grid-like structure using rows and columns.	Places widgets at specific x, y coordinates for precise positioning.
Best For	Simple layouts like stacking widgets vertically or horizontally.	Structured layouts like forms, tables, or grids.	Pixel-perfect layouts or designs requiring absolute control over position.
Ease of Use	Easiest to use, but limited for complex layouts.	Slightly more complex, but intuitive for structured layouts.	Hardest to use due to manual placement and exact positioning.
Resizing Behavior	Widgets auto-adjust based on parent size (`fill`, `expand` options).	Widgets can resize using `sticky` and grid weights for rows/columns.	Widgets do not resize automatically; dimensions must be explicitly defined.

Widgets in Tkinter

The parameters of the "Pack()" method.

Parameter	Description
side	Specifies which side of the parent widget to pack against ("top", "bottom", "left", "right"). Default is "top".
fill	Specifies how the widget should expand to fill the space ("none", "x", "y", "both"). Default is "none".
expand	Boolean (0 or 1) to indicate whether to expand the widget to fill any extra space. Default is 0.
padx	Adds horizontal padding (space) around the widget in pixels. Default is 0.
pady	Adds vertical padding (space) around the widget in pixels. Default is 0.

Widgets in Tkinter

The parameters of the "Grid()" method.

Parameter	Description
row	Specifies the row number in the grid.
column	Specifies the column number in the grid.
rowspan	Number of rows the widget should span.
columnspan	Number of columns the widget should span.
sticky	Defines how the widget should stick to the cell (e.g., N, S, E, W).
padx	Adds horizontal padding around the widget.
pady	Adds vertical padding around the widget.

Widgets in Tkinter

The parameters of the "Place()" method.

Parameter	Description
x	X-coordinate for widget placement (in pixels).
y	Y-coordinate for widget placement (in pixels).
relx	Relative horizontal position (0.0 to 1.0, relative to parent width).
rely	Relative vertical position (0.0 to 1.0, relative to parent height).
width	Absolute width of the widget (in pixels).
height	Absolute height of the widget (in pixels).

Canvas Widget in Tkinter

The Canvas widget lets us display various graphics on the application. It can be used to draw simple shapes to complicated graphs. We can also display various kinds of custom widgets according to our needs.

```
C = Canvas(root, height, width, bd, bg, ..)
```

- root = root window.
- height = height of the canvas widget.
- width = width of the canvas widget.
- bg = background colour for canvas.
- bd = border of the canvas window.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

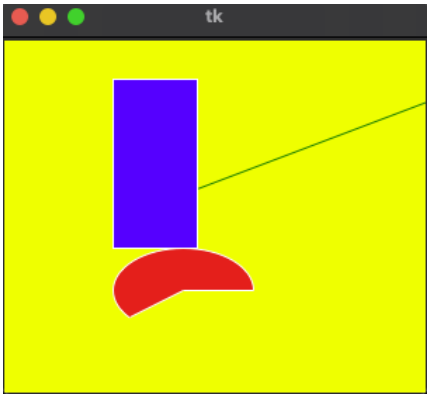
Canvas Widget in Tkinter

Some common drawing methods:

```
from tkinter import *
root = Tk()
C = Canvas(root, bg="yellow", height=250, width=300)
line = C.create_line(108, 120, 320, 40, fill="green")
arc = C.create_arc(180, 150, 80, 210, start=0, extent=220, fill="red")
oval = C.create_rectangle(80, 30, 140, 150,fill="blue")
C.pack()
mainloop()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Canvas Widget in Tkinter



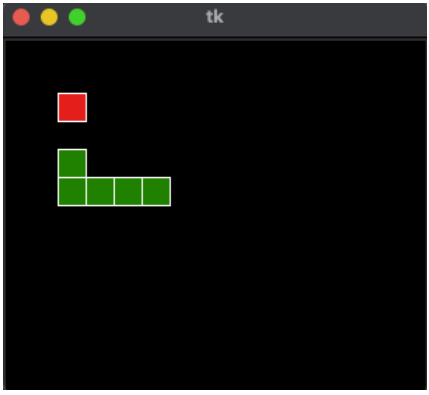
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Canvas Widget in Tkinter

```
from tkinter import *
root = Tk()
canvas = Canvas(root, bg="black", height=250, width=300)
cell_size = 20
snake = [(100, 100), (80, 100), (60, 100), (40, 100), (40, 80)]
food = (40, 40)
for x, y in snake:
    canvas.create_rectangle(x, y, x + cell_size, y + cell_size, fill='green')
canvas.create_rectangle(food[0], food[1], food[0] + cell_size, food[1] + cell_size,
fill='red')
canvas.pack()
mainloop()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Canvas Widget in Tkinter



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Binding function in Tkinter

The binding function is used to deal with the events. We can:

- bind Python's functions and methods to an event.
- bind Python's functions to any particular widget.

Binding function in Tkinter

Binding mouse movement with tkinter Frame.

```
from tkinter import *
from tkinter.ttk import *
root = Tk()
root.geometry('200x100')
def enter(event):
    print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))
frame1 = Frame(root, height = 100, width = 200)
frame1.bind('<Enter>', enter)
frame1.pack()
mainloop()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Binding function in Tkinter

Binding Mouse buttons with Tkinter Frame.

```
from tkinter import *
from tkinter.ttk import *
root = Tk()
root.geometry('200x100')
def double_click(event):
    print('Double clicked at x = % d, y = % d'%(event.x, event.y))
frame1 = Frame(root, height = 100, width = 200)
frame1.bind('<Double 1>', double_click)
frame1.pack()
mainloop()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Binding function in Tkinter

What's the difference between events "<Double 1>", "<Double-Button-1>" and "<Double-Button>"?

<Double 1> and <Double-Button-1> capture only the left click.

<Double-Button> captures both all the (left and right) clicks.

Binding function in Tkinter

Binding keyboard buttons with the root window.

```
from tkinter import *
from tkinter.ttk import *
def key_press(event):
    if event.keysym in ['Left', 'Right', 'Up', 'Down']:
        direction = event.keysym
        print(direction, 'is pressed')
root = Tk()
root.geometry('200x100')
root.bind('<Key>', key_press)
mainloop()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

after() and destroy() functions in Tkinter

Tkinter provides a variety of built-in functions to develop interactive and featured GUI.

- The after() function is also a universal function that can be used directly on the root as well as with other widgets. The function will be run after ms milliseconds.

```
after(parent, ms, function = None, *args)
```

- The destroy() function is a universal widget method i.e we can use this method with any of the available widgets as well as with the main tkinter window.

```
widget_object = Widget(parent, command = widget_class_object.destroy)
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

after() and destroy() functions in Tkinter

```
from tkinter import Tk, mainloop, TOP
from tkinter.ttk import Button
from time import time
root = Tk()
button = Button(root, text = 'Geeks')
button.pack(side = TOP, pady = 5)
print('Running...')
start = time()
root.after(5000, root.destroy)
mainloop()
end = time()
print('Destroyed after % d seconds' % (end-start))
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

after() and destroy() functions in Tkinter

What's the role of .after() and .destroy() functions in the Snanke Game?

A short example:

```
def update_ui():
    snake = [(snake[0][0] + cell_size, snake[0][1])] + snake[:-1]
    canvas.delete('all')
    for x, y in snake:
        canvas.create_rectangle(x, y, x + cell_size, y + cell_size, fill='green')
    root.after(500, update_ui)
root.after(500, update_ui)
```

Check the complete example code named "after_snake_canvas_example.py"!

after() and destroy() functions in Tkinter

Can we replace ".after()" with other functions? for example, the "while" loop and the "time.sleep()" function

- Tkinter's GUI runs in a single thread, and long-running operations can cause the GUI to become unresponsive.
- Using .after() allows the Tkinter main loop (mainloop()) to continue running and keep the interface responsive by processing events and updating the GUI between the scheduled calls.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

after() and destroy() functions in Tkinter

Replacing .after() with a while loop is generally not advisable for several reasons:

- Blocking the Main Loop: A while loop in the main thread of a Tkinter application will block the main event loop (mainloop()), which is responsible for processing user interactions, drawing the GUI, and handling events.
- Lack of Proper Timing: Implementing a timing mechanism in a while loop (e.g., using time.sleep()) to mimic .after() would still block the thread during the sleep time and won't allow for GUI updates or event processing.

Question and Answering (Q&A)