**Introduction to Python Tkinter Module**

**Tkinter** is a **standard library** in python used for creating **Graphical User Interface (GUI)** for Desktop Applications. With the help of **Tkinter** developing **desktop applications** is not a tough task.

The **primary GUI toolkit** we will be using is Tk, which is Python's default GUI library. We'll access Tk from its Python interface called **Tkinter** (short for **Tk interface**).

**Prerequisites for Tkinter**

Before learning **Tkinter** you should have **basic knowledge of Python**.

**GUI Programming in Python**

There are many ways to develop GUI based programs in Python. These different ways are given below:

1. **Tkinter:**

In Python, **Tkinter** is a standard **GUI** (graphical user interface) package. Tkinter is **Python's default GUI module** and also the most common way that is used for **GUI programming** in Python. Note that **Tkinter** is a set of **wrappers**that implement the Tk widgets as Python classes.

1. **wxPython:**

This is basically an open-source, cross-platform **GUI toolkit that is written in C++**. Also an **alternative to Tkinter**.

1. **JPython:**

JPython is a Python platform for Java that is providing Python scripts seamless access to **Java class Libraries** for the local machine.

**What is Tkinter?**

Tkinter in Python helps in **creating GUI**Applications with a minimum hassle. Among various GUI Frameworks, Tkinter is the only framework that is built-in into **Python's Standard Library**.

* An important feature in favor of Tkinter is that it is **cross-platform**, so the same code can easily work on **Windows**, **macOS**, and **Linux**.
* Tkinter is a **lightweight module**.
* It is **simple** to use.

**What are Tcl, Tk, and Tkinter?**

* As mentioned, Tkinter is **Python's default GUI library**, which is nothing but a wrapper module on top of the **Tk toolkit**.
* Tkinter is based upon the Tk toolkit, and which was originally designed for the **Tool Command Language**(Tcl). As Tk is very popular thus it has been ported to a **variety of other scripting languages**, including **Perl (Perl/Tk)**,**Ruby (Ruby/Tk)**, and **Python (Tkinter)**.
* **GUI development portability** and **flexibility of Tk** makes it the right tool which can be used to design and **implement a wide variety of commercial-quality GUI applications.**
* Python**with Tkinter** provides us a **faster and efficient way** in order to build useful applications that would have taken much time if you had to program directly in C/C++ with the help of native OS system libraries.
* Once we have Tkinter up and running, we'll use basic building blocks known as **widgets** to create a variety of desktop applications.

**Install Tkinter**

Chances are, that Tkinter may be already installed on your system along with Python. But it is not true always. So let's first check if it is available.

If you do not have Python installed on your system – Install Python latest Version first, and then check for Tkinter.

You can determine **whether Tkinter is available** for your Python interpreter by attempting to **import the Tkinter module** - If Tkinter is available, then there will be no errors, as demonstrated in the following code:

import tkinter

Nothing exploded, so we know we have **Tkinter available**. If you see any error like module not found, etc, then your**Python interpreter was not compiled with Tkinter enabled**, the module **import fails** and you might need to recompile your **Python interpreter to gain access to Tkinter**.

Adding Tk to your Applications

Basic steps of setting up a GUI application using Tkinter in Python are as follows:

1. First of all, **import the Tkinter module**.
2. The second step is to **create a top-level windowing object** that contains your entire GUI application.
3. Then in the third step, you need to **set up all your GUI components** and their functionality.
4. Then you need to**connect these GUI components** to the underlying application code.
5. Then just enter the main event loop using mainloop()

First Tkinter Example

As mentioned earlier that in GUI programming all main widgets are only built on the top-level window object.

The top-level window object is created by the Tk class in Tkinter.

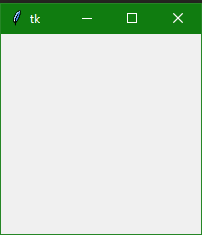
Let us create a top-level window:

import tkinter as tk

win = tk.Tk()

###you can add widgets here

win.mainloop()



Tkinter Methods used above:

The two main methods are used while creating the**Python application** with **GUI**. You must need to remember them and these are given below:

1. **Tk(screenName=None, baseName=None, className='Tk', useTk=1)**

This method is mainly used **to create the main window**. You can also change the**name of the window** if you want, just by changing the **className** to the desired one.

The code used to create the main window of the application is and we have also used it in our above example:

win = tkinter.Tk() ## where win indicates name of the main window object

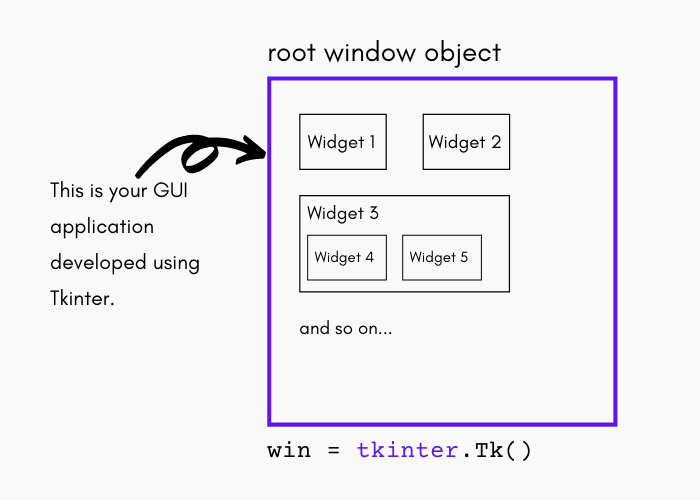
**2. The mainloop() Function**

This method is used to start the application. The mainloop() function is an **infinite loop**which is used to run the application, it will wait for **an event to occur** and **process the event** as long as the window is not closed.

# Tkinter Windows, Widgets and Frames

## Introduction to Tkinter Windows and Widgets

Let's dive in a little deeper, to understand some basics about the Tkinter module and how it works.



* The **top-level window object** in GUI Programming contains all of the **little window objects** that will be part of **your complete GUI**.
* The **little window objects** can be **text labels**, **buttons**, **list boxes**, etc., and these individual little GUI components are known as **Widgets**.
* So, having a top-level window object will act as a container where you will put all your widgets. In Python, you'd typically do so like this using the following code: win = tkinter.Tk()
* The object that is returned by making a call to tkinter.Tk() is usually referred to as **the Root Window**.
* **Top-level windows** are mainly **stand-alone as part of your application**, also**you can have more than one top-level window**for your GUI, but only one of them should be your**root window.**
* First of all, you need to design all your widgets completely, and then **add the real functionality.**
* **The widgets can either be stand-alone**or can be **containers**. If one widget contains other widgets, it is considered the**parent of those widgets**.
* Similarly, if **a widget is contained within another widget**, it's known as**a child of the parent,** the**parent is the next immediate** enclosing container widget.
* The **widgets also have some associated behaviors**, such as **when a button is pressed**, or**text is filled into a text field**, so we have events attached to these actions.
* The behavior of widgets generates events, and the **GUI's response to events** is known as **Callbacks** - because **they 'call' a function just to handle the event**that occurred.

## Tkinter Event-Driven Processing

In Tkinter, we have windows and widgets coming together to form a GUI application. But the GUI application is just the frontend of the application. You would want to execute some code logic when the end-user uses those widgets.

Whenever an action is performed on any widget, an event is generated, which we can handle to perform any operation.

* Events(behavior of widgets) can include **pressing a button**, **movement of the mouse**, **hitting the return or Enter key**, **gaining or losing 'focus'**, etc.
* The **entire system of events** that occurs from the**beginning until the end of any GUI application** is what drives it and hence it is also known as E**vent-Driven Processing**.
* Let us take a **simple mouse movement** example: Suppose that the**pointer of the mouse is just sitting somewhere** on top of your GUI application. If you will move the mouse to another part of your application, something has to cause the movement of the mouse to be replicated by the cursor on your screen(on top of your GUI application). These are **'cursor move'** events that the system **must process to portray** your cursor moving across the window. At the time you will **stop moving the mouse**, no more events need to be processed, so everything **just stays still on the screen again**.

Here are some basic definitions through which you will be able to understand the concept of Windows, widgets, and frames in Tkinter.

## Tkinter Windows

The term "Window" has different meanings in the different contexts, But generally **"Window"**refers to a rectangular area somewhere on the user's display screen through which you can interact.

Then there comes the concept of Top Level Window in Tkinter.

## Tkinter Top-Level Window

The Top-Level Window is a window that **exists independently on the screen**. You can decorate the top-level window with the standard frame and controls for the desktop manager. It can usually be moved **around the desktop, and also you can resize it if you want to do so**.

Then there comes the concept of Widgets.

## Tkinter Widgets

The term "Widgets" is a generic term that refers to the**building blocks that make up an application in a graphical user interface**.

Let us list out the core widgets with their categories:

* **Container**

Under this category, the widgets that lies are frame, labelframe, toplevel, and paned window.

* **Buttons**

Under the category of Buttons, there are buttons, radiobuttons, checkbuttons (checkbox), and menubuttons.

* **Text Widgets**

Under the category of text widgets, there are labels, messages, text.

* **Entry Widgets**

Under this category, the widgets are scale, scrollbar, Listbox, slider, spinbox, entry (single-line), optionmenu, text (multiline), and canvas (vector and pixel graphics).

## Tkinter Frames

A frame is basically a rectangular area that can contain other widgets. In Tkinter, there is a Frame widget that is the basic unit of organization for complex layouts. It is a widget which has no special styling or GUI component of its own. It is just used to hold other Tkinter widgets in case of complex GUI layouts.

**Note:** It is important to note here that whenever any widget is created, there is also a parent-child relationship that is created. Just take an example, if you place a button inside a frame, the frame widget is called the parent of the Button widget.

## Tkinter Basic Example

Let us take an example where we will create a Tkinter application with a simple Text widget:

from tkinter import \*

win = Tk() # Create the root (base) window

win.geometry("200x200")

w = Label(win, text="Hello, Welcome to Talent Battle!") # Create a label with words

w.pack() # Put the label into the window

win.mainloop()# Start the event loop

Tkinter Windows

The Tkinter window is the **foundational element** of the Tkinter GUI. Tkinter window is a**container in which all other GUI elements(widgets) live**.

Here is the **syntax** for creating a basic Tkinter Window:

win = Tk()

Yes, we use the Tk() function to create our Tkinter app window in which all the other components are added.

Tkinter Windows Example:

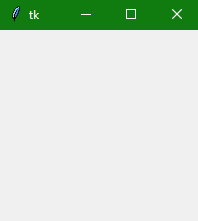
Here is a simple example,

from tkinter import \*

win = Tk()

# run the app window

win.mainloop()



In the above example, the mainloop() function is used to run the GUI application.

Tkinter Customized Window

Let us now cover a basic example where we will create a Basic GUI Application using properties like **title** and **geometry**.

Here we have the code to demonstrate the steps used in the creation of a customized Tkinter Window:

from tkinter import \*

window = Tk()

# You can add your widgets here

window.title('Hello Talent Battle')

window.geometry("300x200+10+20")

window.mainloop()

Here is what we have done in the code:

* The first step is to import the **Tkinter module in the code.**
* After importing, the second step is to set up the application object by calling the Tk() function. This will create a top-level window (root) having a frame with a**title bar**and **control box** with the minimize and close buttons, and a**client area to hold other widgets**.
* After that, you can add the widgets you might want to add in your code like buttons, textbox, scrollbar, labels, and many more.
* The window.title() function is used to provide the title to the user interface as you can see in the output.
* In the line window.geometry("300x200+10+20); the geometry() method defines the **width**, **height,** and **coordinates** of the **top left corner** of the frame as follows (all values are generally in pixels) in the same way. Here is the syntax:

window.geometry("widthxheight+XPOS+YPOS")

* After that, the application object enters an **event listening loop** by calling the mainloop() method. In this way, the application is now **constantly waiting for any event generated** on the elements in it. There could be an event like **text entered in a text field**, a **selection made from the dropdown** or radio button, single/double click actions of the mouse, etc.

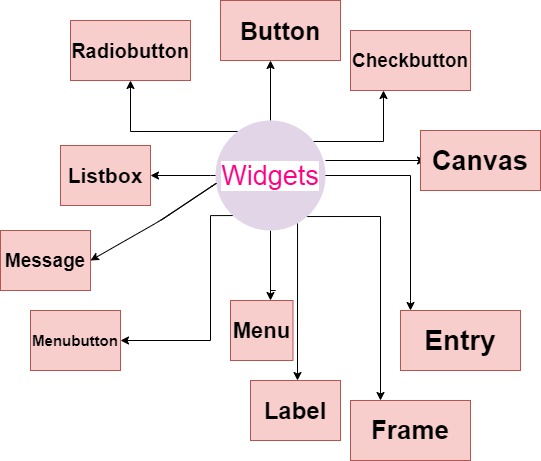
The application's functionality involves **executing appropriate callback functions** in response to a particular type of event.

The event loop will terminate as soon as there is a click made on the **close button on the title bar**.

Python Tkinter Widgets

Tkinter Widgets

There are various controls, such as **buttons**, **labels**, **scrollbars**, **radio buttons**, and **text boxes** used in a GUI application. These**little components** or controls of **Graphical User Interface (GUI)** are known as **widgets** in Tkinter.



These are**19 widgets** available in Python Tkinter module. Below we have all the widgets listed down with a basic description:

| **Name of Widget** | **Description** |
| --- | --- |
| Button | If you want to add a button in your application then **Button widget** will be used. |
| Canvas | To draw a complex layout and pictures (like graphics, text, etc.)**Canvas Widget** will be used. |
| CheckButton | If you want to display a number of options as checkboxes then **Checkbutton widget** is used. It allows you to select multiple options at a time. |
| Entry | To display a **single-line text field** that accepts values from the user **Entry widget** will be used. |
| Frame | In order to group and organize another widgets Frame widget will be used. Basically it acts as**a container that holds other widgets**. |
| Label | To Provide a **single line caption**to another widget Label widget will be used. It can **contain images**too. |
| Listbox | To provide a user with a list of options the**Listbox widget** will be used. |
| Menu | To provides commands to the user**Menu widget** will be used. Basically these **commands are inside the Menubutton**. This widget mainly creates**all kinds of Menus**required in the application. |
| Menubutton | The **Menubutton widget** is used to display the menu items to the user. |
| Message | The**message widget** mainly displays a message box to the user. Basically it is a **multi-line text which is non-editable.** |
| Radiobutton | If you want the number of options to be displayed as radio buttons then the**Radiobutton widget** will be used. You can select one at a time. |
| Scale | **Scale widget** is mainly a**graphical slider** that allows you to select values from the scale. |
| Scrollbar | To scroll the window up and down the **scrollbar widget** in python will be used. |
| Text | The **text widget**mainly provides a **multi-line text field** to the user where users and enter or edit the text and it is different from Entry. |
| Toplevel | The **Toplevel widget**is mainly used to provide us with a separate window container |
| SpinBox | The **SpinBox** acts as an entry to the "**Entry widget**" in which value can be input just by **selecting a fixed value of numbers**. |
| PanedWindow | The**PanedWindow** is also a **container widget** that is mainly used to **handle different panes**. Panes arranged inside it can **either Horizontal or vertical** |
| LabelFrame | The **LabelFrame widget**is also a container widget used to mainly handle the complex widgets. |
| MessageBox | The **MessageBox** **widget**is mainly used to display messages in the Desktop applications. |

**Reading Images With Python – Tkinter**

There are numerous tools for designing GUI (Graphical User Interface) in Python such as tkinter, wxPython, JPython, etc where Tkinter is the standard Python GUI library, it provides a simple and efficient way to create GUI applications in Python.

**Reading Images With Tkinter**

In order to do various operations and manipulations on images, we require Python Pillow package. If the Pillow package is not present in the system then it can be installed using the below command.

* In Command prompt:

pip install Pillow

* In Anaconda prompt:

conda install -c anaconda pillow

**Example 1:** The below program demonstrates how to read images with tkinter using PIL.

|  |
| --- |
| # importing required packages  **import** tkinter  **from** PIL **import** ImageTk, Image  **import** os    # creating main window  root **=** tkinter.Tk()    # loading the image  img **=** ImageTk.PhotoImage(Image.open("sample.jpeg"))    # reading the image  panel **=** tkinter.Label(root, image **=** img)    # setting the application  panel.pack(side **=** "bottom", fill **=** "both",             expand **=** "yes")    # running the application  root.mainloop() |

In the above program, an image is loaded using the PhotoImage() method and then it is read by using the Label() method.The pack() method arranges the main window and the mainloop() function is used to run the application in an infinite loop.

**Example 2:** Let us look at another example where we arrange the image parameters along with application parameters.

|  |
| --- |
| # importing required packages  **import** tkinter  **from** PIL **import** ImageTk, Image    # creating main window  root **=** tkinter.Tk()    # arranging application parameters  canvas **=** tkinter.Canvas(root, width **=** 500,                          height **=** 250)    canvas.pack()    # loading the image  img **=** ImageTk.PhotoImage(Image.open("sample.ppm"))    # arranging image parameters  # in the application  canvas.create\_image(135, 20, anchor **=** NW,                     image **=** img)    # running the application  root.mainloop() |

In the above program, the application parameters are handled by using the Canvas() method and the image parameters are handled using create\_image() method such that the image gfg.ppm is displayed in the main window having defined height and width.

**Note:** The Canvas method create\_image(x0,y0, options …) is used to draw an image on a canvas. create\_image doesn’t accept an image directly. It uses an object which is created by the PhotoImage() method. The PhotoImage class can only read GIF and PGM/PPM images from files.

**Loading Images in Tkinter using PIL**

In this article, we will learn how to load images from user system to Tkinter window using PIL module. This program will open a dialogue box to select the required file from any directory and display it in the tkinter window.

**Install the requirements –**  
**Use this command to install Tkinter :**

pip install python-tk

**Use this command to install PIL :**

pip install pillow

**Importing modules –**

|  |
| --- |
| **from** tkinter **import** **\***    # loading Python Imaging Library  **from** PIL **import** ImageTk, Image    # To get the dialog box to open when required  **from** tkinter **import** filedialog |

**Note:** The ImageTk module contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images and filedialog is used for the dialog box to appear when you are opening file from anywhere in your system or saving your file in a particular position or place.   
    
**Function to create a Tkinter window consisting of a button –**

|  |
| --- |
| # Create a window  root **=** Tk()    # Set Title as Image Loader  root.title("Image Loader")    # Set the resolution of window  root.geometry("550x300 + 300 + 150")    # Allow Window to be resizable  root.resizable(width **=** True, height **=** True)    # Create a button and place it into the window using grid layout  btn **=** Button(root, text **=**'open image', command **=** open\_img).grid(                                          row **=** 1, columnspan **=** 4)  root.mainloop() |

The Button object is created with text ‘open image’. On clicking it the open\_image function will be invoked. 

**Function to place the image onto the window –**

|  |
| --- |
| **def** open\_img():      # Select the Imagename  from a folder      x **=** openfilename()        # opens the image      img **=** Image.open(x)        # resize the image and apply a high-quality down sampling filter      img **=** img.resize((250, 250), Image.ANTIALIAS)        # PhotoImage class is used to add image to widgets, icons etc      img **=** ImageTk.PhotoImage(img)        # create a label      panel **=** Label(root, image **=** img)        # set the image as img      panel.image **=** img      panel.grid(row **=** 2) |

The openfilename function will return the file name of image. 

**Function to return the file name chosen from a dialog box –**

|  |
| --- |
| **def** openfilename():        # open file dialog box to select image      # The dialogue box has a title "Open"      filename **=** filedialog.askopenfilename(title **=**'sample')  **return** filename |

To run this code, save it by the extension .py and then open cmd (command prompt) and move to the location of the file saved and then write the following

python "filename".py

and press enter and it will run. Or can be run directly by simply double-clicking your .py extension file.