Revised: 7/27/2020

Graphical User Interfaces

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- GUIs and Event-Driven Programming
- Windows
- Labels
- Frames
- Dialog Boxes
- Buttons
- Entry Fields
- StringVars
- Check Buttons
- Radio Buttons
- Scales

Colors/Fonts

- Global Variable Names – Brown
- Local Variable Names – Lt Blue
- Literals – Blue
- Keywords – Orange
- Operators/Punctuation – Black
- Functions – Purple
- Parameters – Gold
- Comments – Gray
- Modules – Pink
- Object/Class Names – Green

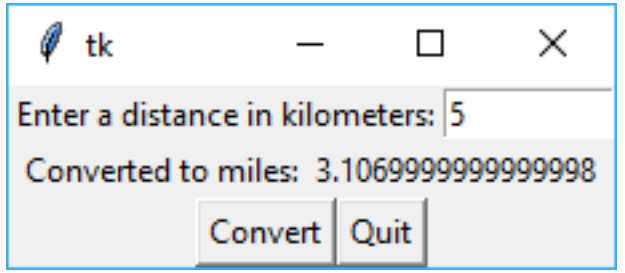
Source Code – **Consolas**
Output – Courier New

Graphical User Interfaces

- A ***graphical user interface*** (*GUI* or "*gooey*") allows a user to interact with a program using pictures, icons and other visual components.
  - As opposed to programs using a *command line interface* (*CLI*).

```
========== RESTART: C:/testing/kiloprogram.py =
Enter a distance in kilometers: 5
Converted to miles: 3.1069999999999998
>>>
```

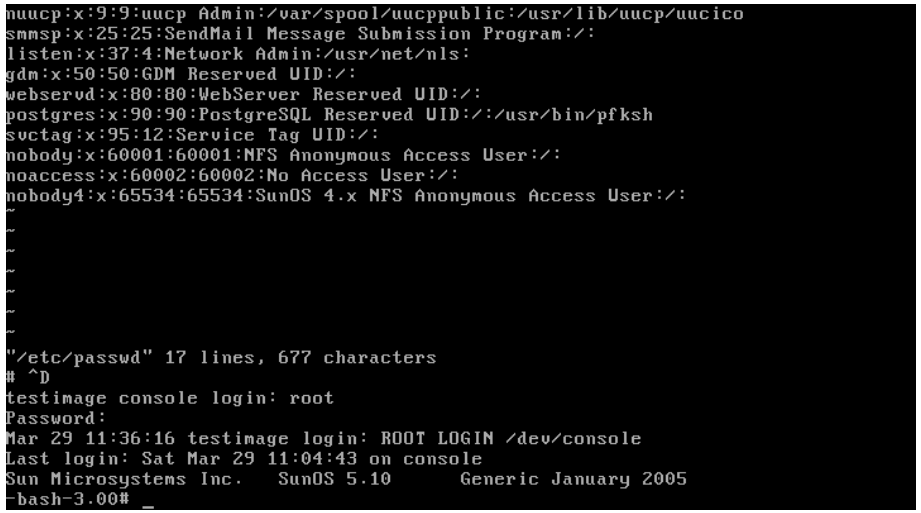
Command Line Interface



Graphical User Interface

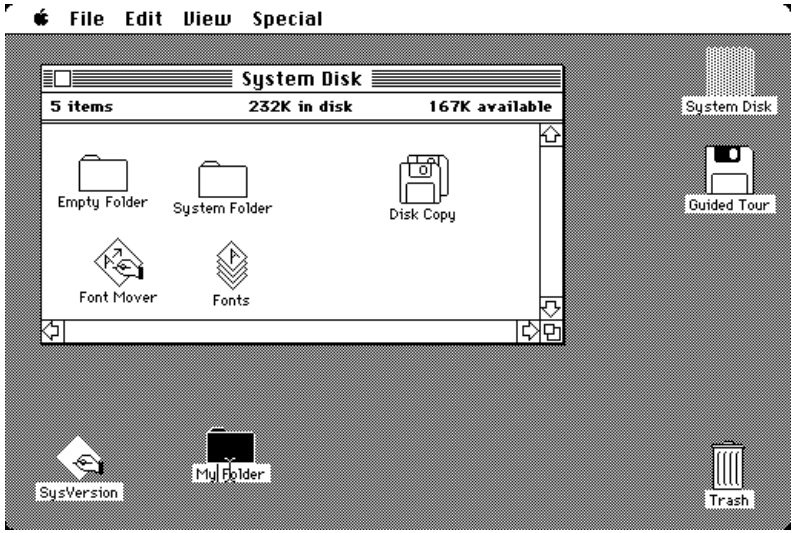
Graphical User Interfaces

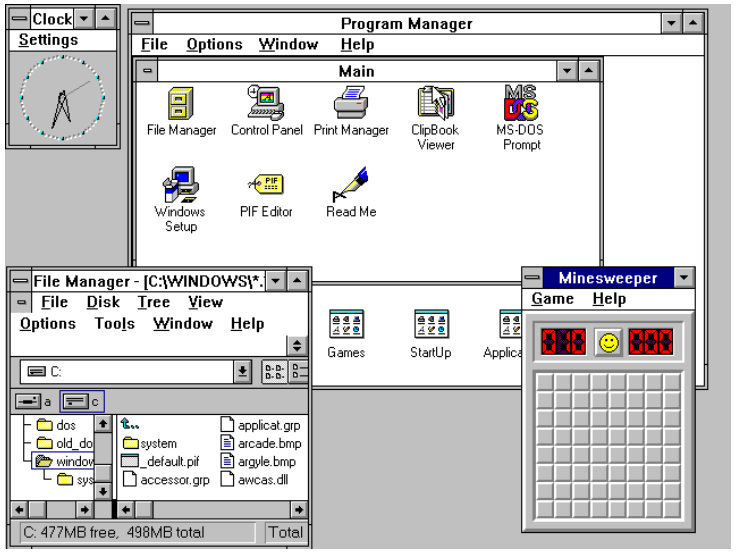
- Prior to GUIs, all work on a computer was done using a CLI.
  - This made it very difficult for new computers users.
  - Commands needed to be memorized and entered using a keyboard.



Graphical User Interfaces

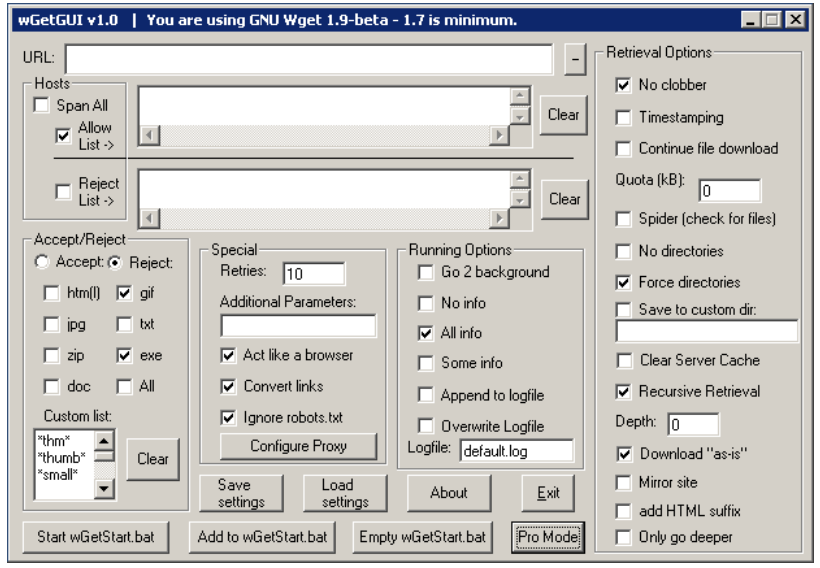
- With a GUI, users did not need to remember commands to use the operating system.
  - They could use a mouse to click buttons to perform the commands and click icons to open programs.



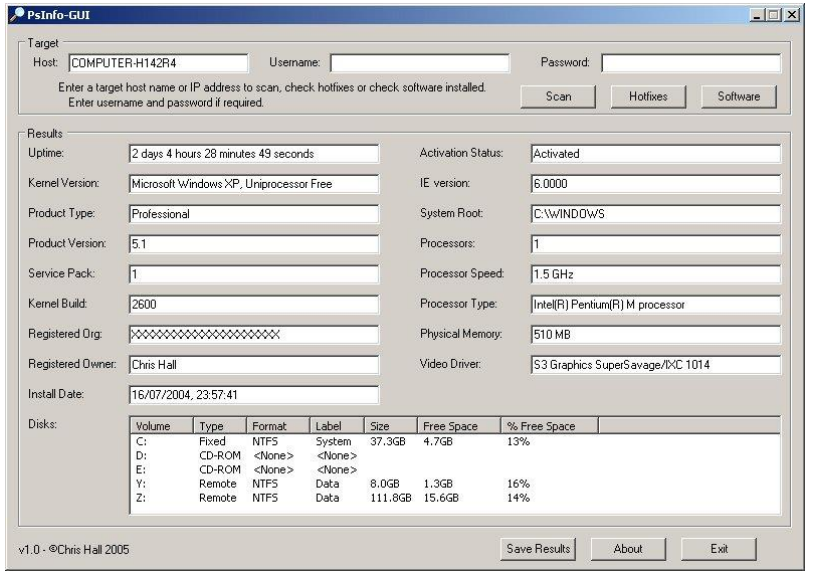


Graphical User Interfaces

- A successful GUI is one that is *user-friendly*.
  - The interface is intuitive, organized and familiar.
  - The user should not feel overwhelmed.



Cluttered GUI



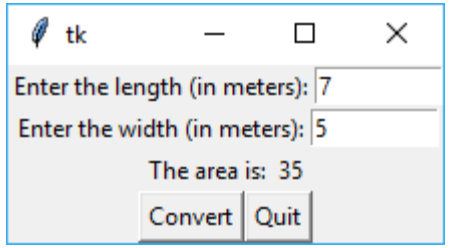
Clean GUI

Graphical User Interfaces

- Events in a command line program are predetermined.
  - The user enters a length, then enters a width and the area is printed.

```
Enter the length (in meters): 7
Enter the width (in meters): 5
The area is 35 meters.
>>>
```

- In a graphical program, the user determines the order of events.
  - The user could enter the length first, or the width first.
  - The area is not displayed until the user clicks the Convert button



Graphical User Interfaces

- GUI programs are *event-driven*.
  - The user *triggers* events in the program to happen.
    - Clicking buttons.
    - Checking check boxes.
    - Selecting an item from a menu or list.

- A GUI program, essentially, exists in a loop.
  - The loop keeps the GUI alive/running and waits for events to occur.
  - It then executes the code associated with a particular event.

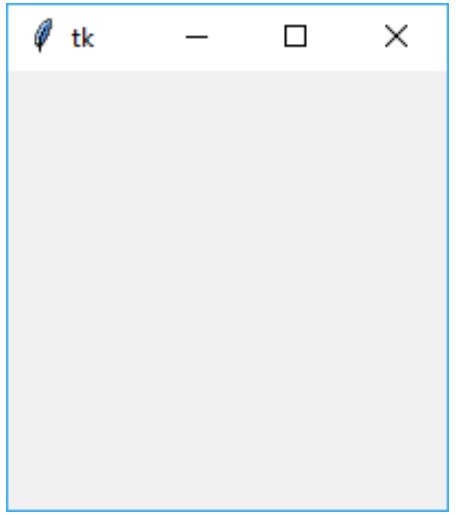
The tkinter module

- Python does not have built-in GUI capabilities.
- The tkinter module (which comes with Python) allows us to create graphical user interfaces.
  - Short for “Tk interface”
  - Other languages use the Tk framework for creating windowed applications.
- There are other GUI-development libraries for Python.

Creating a Window

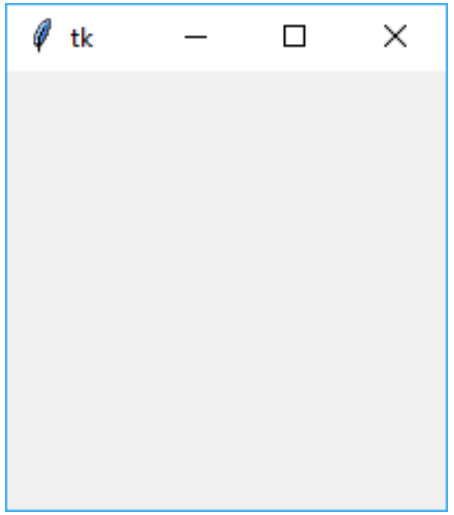
- A ***window*** is the general term for a rectangular container of graphical components.

- Windows are normally decorated with a title and minimize/maximize/close buttons.



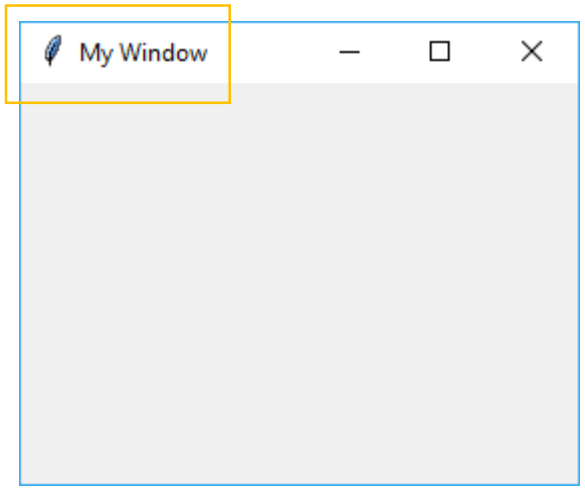
Creating a Window

```
import tkinter

def main() :
    #Creates the window
    test_window = tkinter.Tk()

    #Enters the main loop, displaying the window
    #and waiting for events
    tkinter.mainloop()

main()
```



Setting the Window's Title

```
import tkinter

def main() :
    #Creates the window
    test_window = tkinter.Tk()

    #Sets the window's title
    test_window.wm_title("My Window")

    #Enters the main loop, displaying the window
    #and waiting for events
    tkinter.mainloop()

main()
```



Widgets

- A ***widget*** is the general term for a graphical component the user interacts with.
  - Buttons, checkboxes, and entry fields are all examples of widgets.

- A ***label*** is a widget that displays text or an image.
  - The text is not editable by a user.

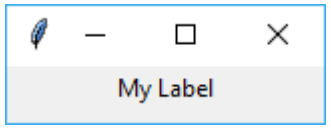
Creating a Label

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_label = tkinter.Label(test_window, text="My Label")

    tkinter.mainloop()

main()
```

Creating a Label

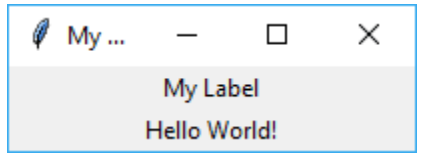
- Calling a widget's pack function, makes it visible in the window.
  - All widgets must have their pack functions called or else the widget will not be added.

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_label = tkinter.Label(test_window, text="My Label")

    test_label.pack()

    tkinter.mainloop()

main()
```



Creating a Label

- Adds a second label to the window

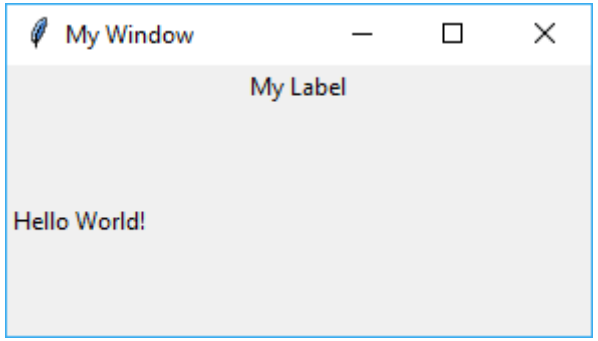



```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_label = tkinter.Label(test_window, text="My Label")
    test_label2 = tkinter.Label(test_window, text="Hello World!")

    test_label.pack()
    test_label2.pack()

    tkinter.mainloop()

main()
```

Creating a Label

- The pack function can specify the widget's orientation.
  - Valid arguments at "right", "left", "bottom" and "top"

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_label = tkinter.Label(test_window, text="My Label")
    test_label2 = tkinter.Label(test_window, text="Hello World!")

    test_label.pack(side="top")
    test_label2.pack(side="left")

    tkinter.mainloop()

main()
```



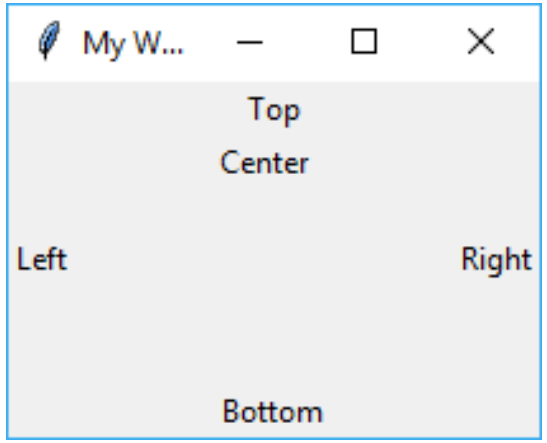
Adding/Packing Widgets

- Widgets are always added in the order they are packed.

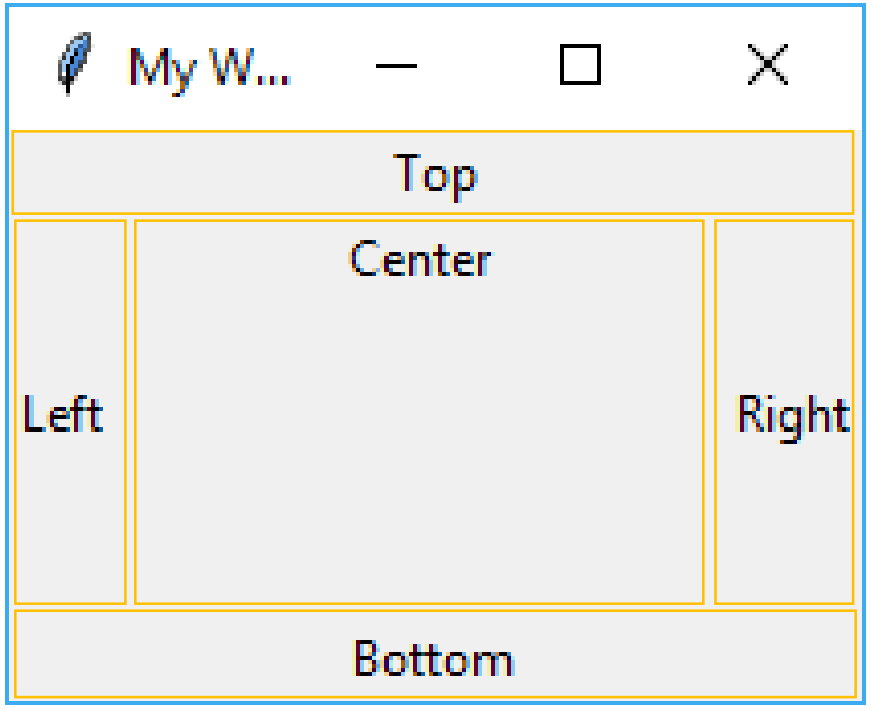
```
test_window = tkinter.Tk()
test_window.wm_title("My Window")
test_label = tkinter.Label(test_window, text="My Label")
test_label2 = tkinter.Label(test_window, text="Hello World!")
test_label3 = tkinter.Label(test_window, text="Nice to meet you!")

test_label.pack(side="top")
test_label2.pack(side="left")
test_label3.pack(side="left")

tkinter.mainloop()
```

Adding/Packing Widgets

```
test_window = tkinter.Tk()
test_window.wm_title("My Window")
test_label = tkinter.Label(test_window, text="Top")
test_label2 = tkinter.Label(test_window, text="Bottom")
test_label3 = tkinter.Label(test_window, text="Left")
test_label4 = tkinter.Label(test_window, text="Right")
test_label5 = tkinter.Label(test_window, text="Center")

test_label.pack(side="top")
test_label2.pack(side="bottom")
test_label3.pack(side="left")
test_label4.pack(side="right")
test_label5.pack()

tkinter.mainloop()
```



Adding/Packing Widgets



Frames

- This “border layout” places some limits on how we can display widgets directly in a window.
  - This is why widgets are rarely packed on the window directly.

- A ***Frame*** is a widget that contains widgets.

- We can add widgets how we want to individual frames.
  - We can then pack the frames in the center of the window.

Creating a Frame

- When creating a Frame, we must specify the window it belongs to.

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    upper_frame = tkinter.Frame(test_window)
    lower_frame = tkinter.Frame(test_window)


main()
```

Creating a Frame

- Then, we can add widgets to the desired Frame.

```
upper_frame = tkinter.Frame(test_window)
lower_frame = tkinter.Frame(test_window)

label1 = tkinter.Label(upper_frame, text="Label 1")
label2 = tkinter.Label(upper_frame, text="Label 2")

labelA = tkinter.Label(lower_frame, text="Label A")
labelB = tkinter.Label(lower_frame, text="Label B")
```

Creating a Frame

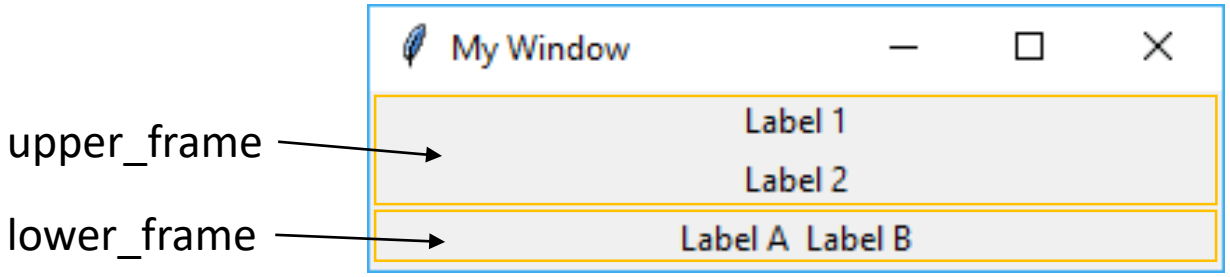
- Pack the widgets with the desired orientation/side.
  - The side is relative to its Frame's side, not the window's.

```
label1 = tkinter.Label(upper_frame, text="Label 1")
label2 = tkinter.Label(upper_frame, text="Label 2")

labelA = tkinter.Label(lower_frame, text="Label A")
labelB = tkinter.Label(lower_frame, text="Label B")

label1.pack(side="top")
label2.pack(side="top")
labelA.pack(side="left")
labelB.pack(side="left")
```

Creating a Frame

- Pack the Frames with the desired orientation/side.
  - The side is relative to the window.
- Start the main loop.

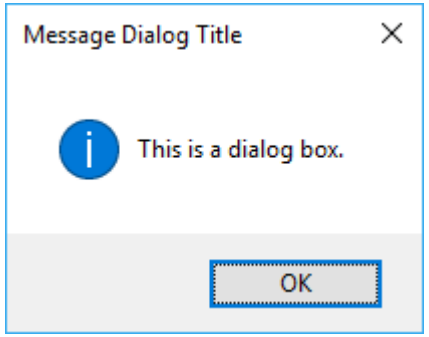
```
label1.pack(side="top")
label2.pack(side="top")
labelA.pack(side="left")
labelB.pack(side="left")

upper_frame.pack()
lower_frame.pack()

tkinter.mainloop()
```



Dialog Boxes

- A ***dialog box*** is a small window that usually displays an informational message to a user and is accompanied with an “OK” or “Close” button.

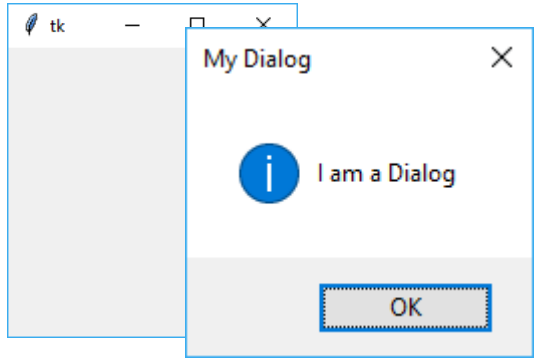
- The tkinter.messagebox module can be used to create dialog boxes.



Dialog Boxes

- Dialog boxes require a root window.
  - If one doesn't exist, one will be created.
  - For this reason, dialogs aren't common in Python CLI applications.

```
import tkinter.messagebox

def main() :
    test_window = tkinter.messagebox.showinfo("My Dialog",
                                              "I am a Dialog")

main()
```

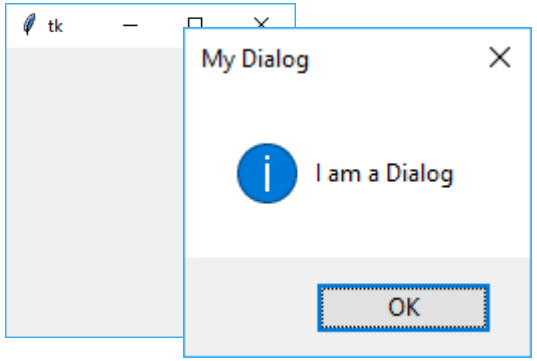


Dialog Boxes

- Dialog boxes are *modal*.
  - This means, while they are open, they pause the application until the OK button is pressed.

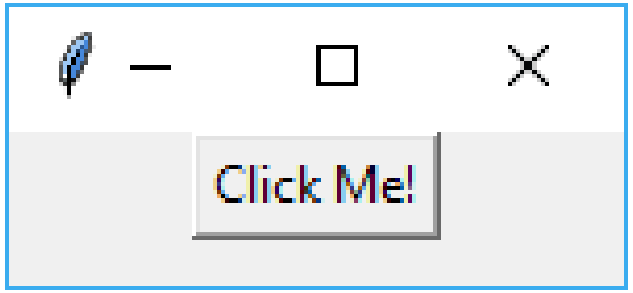
```
import tkinter.messagebox

def main() :
    tkinter.messagebox.showinfo("My Dialog", "I am a Dialog")
    print("Dialog is closed.")

main()
```

Will not print to the console until the dialog is closed.



Buttons

- A ***button*** is the general term for a rectangular component the user presses with the mouse cursor.
  - Some event typically occurs when the button is pressed.



Creating a Button

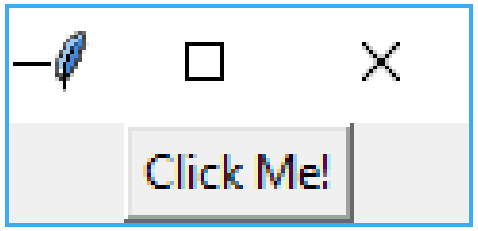
- When creating a button, we must specify
  - The window or frame it belongs to.
  - The text displayed on the button.
  - The function called when the button is pressed. NO PARENTHESES!

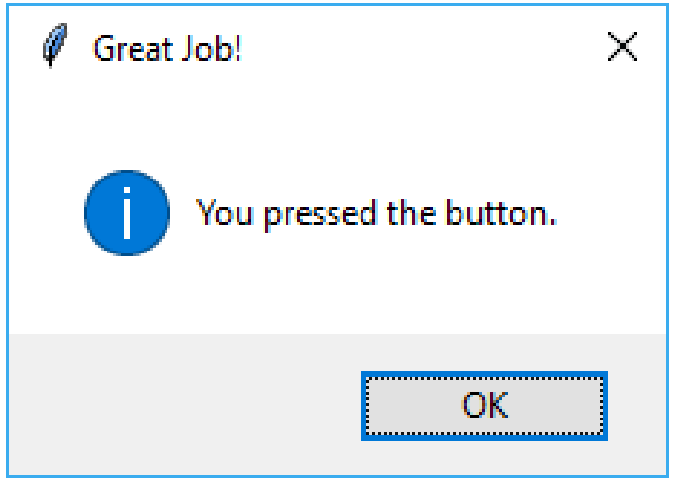
```
test_button = tkinter.Button(test_window,
                             text="Click Me!",
                             command=showdialog)
```

Creating a Button

```
import tkinter
import tkinter.messagebox

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_button = tkinter.Button(test_window,
                                 text="Click Me!",
                                 command=showdialog)


    test_button.pack()
    tkinter.mainloop()

def showdialog() :
    tkinter.messagebox.showinfo("Great Job!", "You pressed the button.")

main()
```

Creating a Button





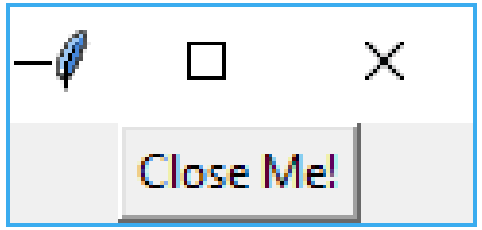
Creating a Quit Button

- The action to close (“destroy”) a window is to call it’s destroy function.
  - We can make this be the action for a regular button to perform.

```
test_button = tkinter.Button(test_window,
                             text="Close Me!",
                             command= test_window.destroy)
```

Creating a Quit Button

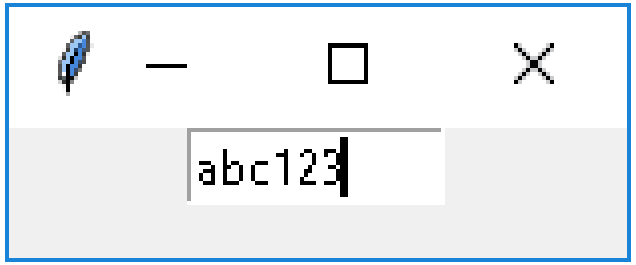
```
import tkinter
import tkinter.messagebox

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_button = tkinter.Button(test_window,
                                 text="Close Me!",
                                 command= test_window.destroy)


    test_button.pack()
    tkinter.mainloop()

main()
```



Entry Fields

- An ***entry field*** (or ***text field***) is the general term for a rectangular component the user types information into.

- The Entry widget is used to create an entry field.



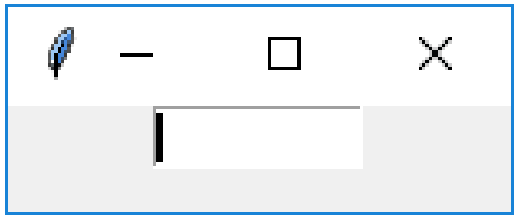
Creating an Entry Field

- When creating an entry field, we must specify
  - The window or frame it belongs to.
  - The width of the entry field.

```
test_entry = tkinter.Entry(test_window, width=10)
```

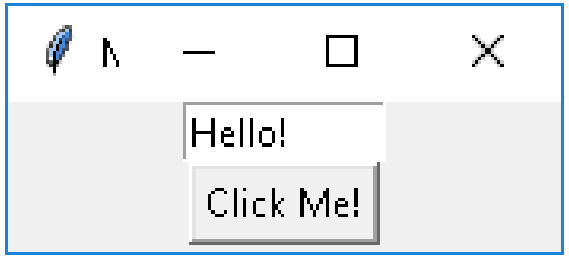
Creating an Entry Field

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_entry = tkinter.Entry(test_window, width=10)

    test_entry.pack()
    tkinter.mainloop()

main()
```



Retrieving an Entry Field's value

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    global test_entry
    test_entry = tkinter.Entry(test_window, width=10)
    test_button = tkinter.Button(test_window,
                                 text="Click Me!",
                                 command=showdialog)
    test_entry.pack(side="top")
    test_button.pack(side="top")
    tkinter.mainloop()

def showdialog() :
    tkinter.messagebox.showinfo("Your text", test_entry.get())

main()
```

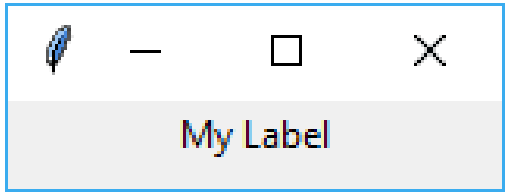


Using Labels to Display Output

- When a label is created using a string literal, its value/text can't be changed.

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    test_label = tkinter.Label(test_window, text="My Label")

    test_label.pack()

    tkinter.mainloop()

main()
```

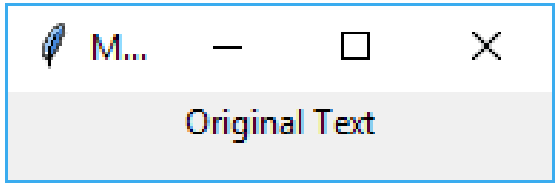
Can't be changed later



Using Labels to Display Output

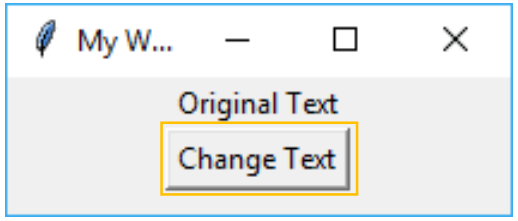
- Using a **StringVar** object (provided by tkinter) allows us to make labels with text that can be changed.

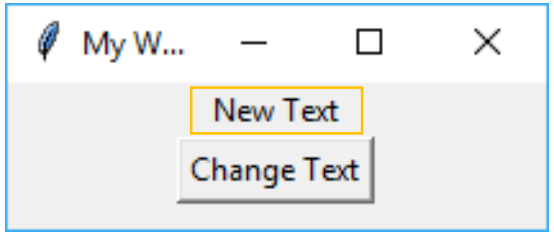
```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    global label_text
    label_text = tkinter.StringVar()
    label_text.set("Original Text")
    test_label = tkinter.Label(test_window, textvariable=label_text)
    test_label.pack()

    tkinter.mainloop()

main()
```



Using Labels to Display Output

```
import tkinter

def main() :
     test_window = tkinter.Tk()
     test_window.wm_title("My Window")
     global label_text
     label_text = tkinter.StringVar()
     label_text.set("Original Text")
     test_label = tkinter.Label(test_window, textvariable=label_text)
     test_button = tkinter.Button(test_window,
                                  text="Change Text",
                                  command=changeText)
     test_label.pack()
     test_button.pack()
     tkinter.mainloop()

def changeText() :
     label_text.set("New Text")

main()
```

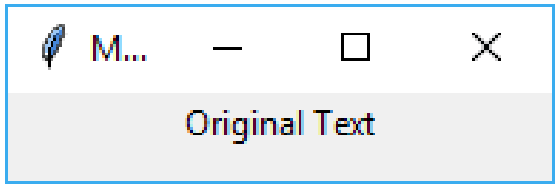




StringVars and Entry Fields

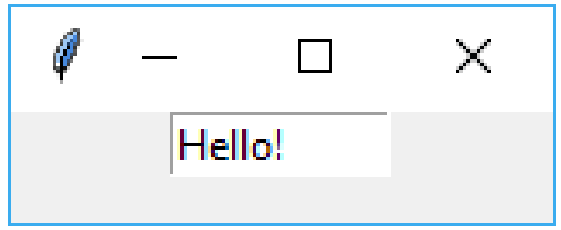
- Using a **StringVar** object (provided by tkinter) with an entry field allows us to get and set the text displayed in the field.

```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    global label_text
    label_text = tkinter.StringVar()
    label_text.set("Original Text")
    test_label = tkinter.Label(test_window, textvariable=label_text)
    test_label.pack()

    tkinter.mainloop()

main()
```



StringVars and Entry Fields

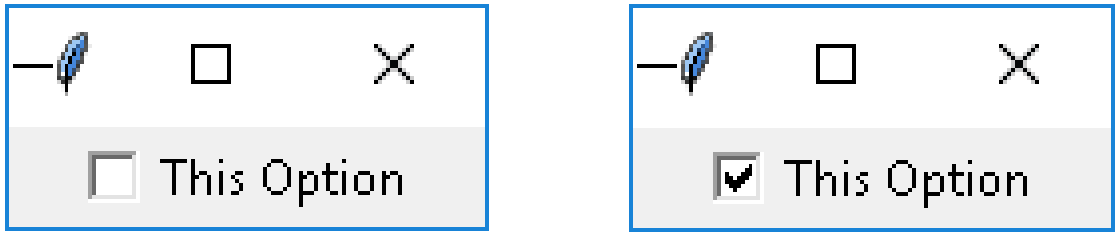
```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    entry_text = tkinter.StringVar()
    entry_text.set("Hello!")
    test_entry = tkinter.Entry(test_window, textvariable= entry_text, width=10)
    test_entry.pack(side="top")
    tkinter.mainloop()

main()
```



Check Buttons

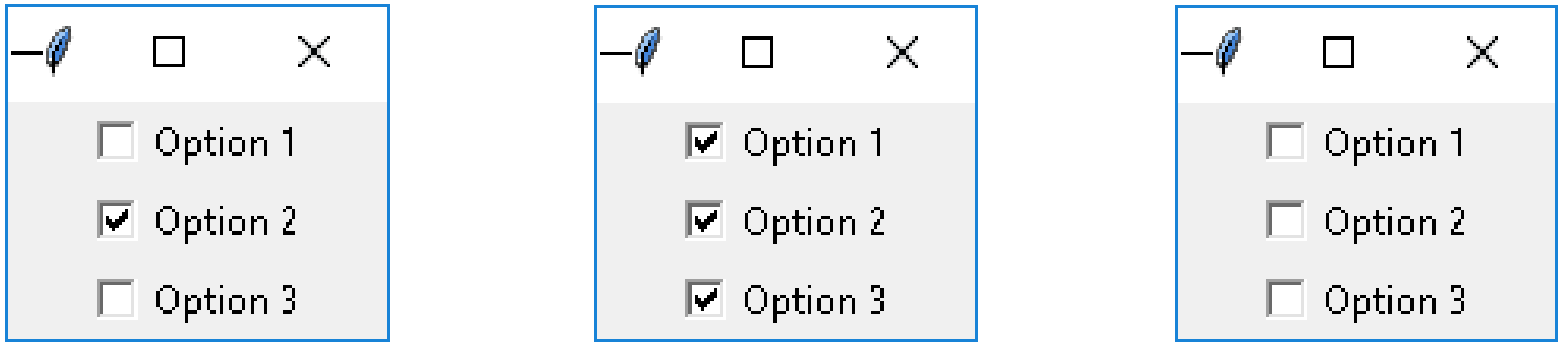
- A ***check button*** (or check box) is a square component the user presses with the mouse cursor to make a selection.
  - A “check mark” appears in the square.
  - Includes an accompanying text label.

- Normally used for yes/no or on/off user selections.



Check Buttons

- Check buttons may appear alone but often appear in groups.
- The normal implementation of check buttons will allow the user to select any number of the options.



IntVars

- To determine the state of a check button, we must use an IntVar object.

- The check button will set its value to
  - 0 when the check button is not selected.
  - 1 when the check button is selected.

- Each check button needs its own IntVar associated with it.

```
global cbvar
cbvar = tkinter.IntVar()
cbvar.set(0)
```

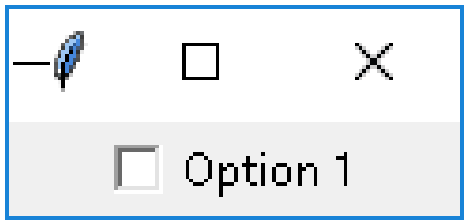
Creating a Check Button

- When creating a check button, we must specify
  - The window or frame it belongs to.
  - The text displayed next to the check button.
  - The IntVar associated with the check button.

```
global cbvar
cbvar = tkinter.IntVar()
cbvar.set(0)

test_cbutton = tkinter.Checkbutton(test_window,
                                   text="Option 1",
                                   variable=cbvar)
```

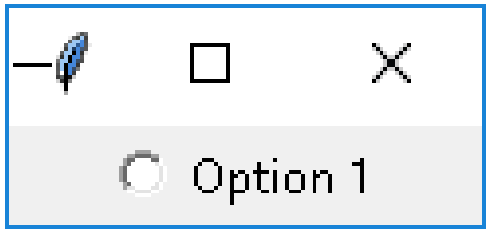
Creating a Check Button

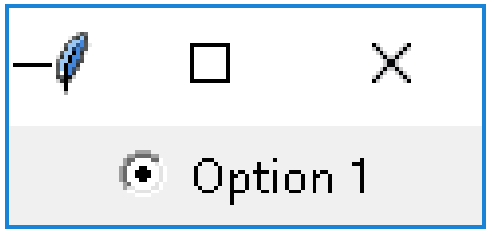
```
import tkinter

def main() :
    test_window = tkinter.Tk()
    test_window.wm_title("My Window")
    global cbvar
    cbvar = tkinter.IntVar()
    cbvar.set(0)

    test_cbutton = tkinter.Checkbutton(test_window,
                                       text="Option 1",
                                       variable=cbvar)
    test_cbutton.pack()
    tkinter.mainloop()

main()
```



Radio Buttons

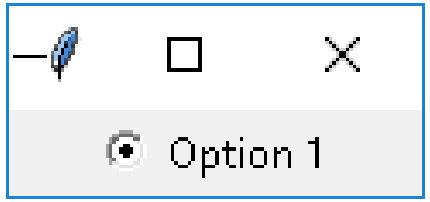
- A ***radio button*** is a circular component the user presses with the mouse cursor to make a selection.
  - A “dot” appears in the circle.
  - Includes an accompanying text label.



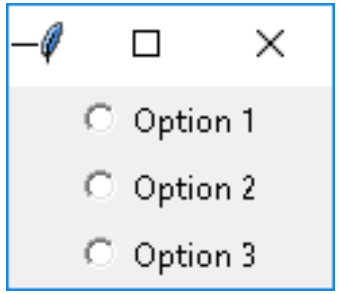


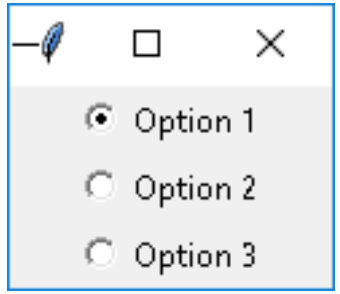
Radio Buttons

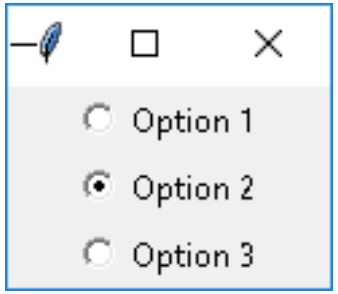
- Unlike check buttons, radio buttons cannot be unselected.

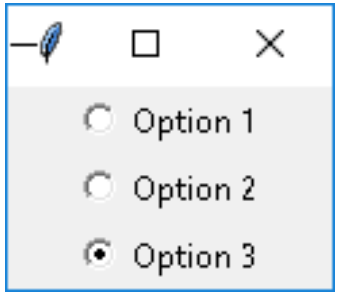


- Radio buttons almost always appear in groups.
  - The user then chooses one option from the group.









Radio Buttons

- To determine the state of a radio button, IntVars are used.
- The group of radio buttons all share the same IntVar.
- Each radio button has its own unique value.
  - Unlike a check button where its IntVar is either 0 or 1.

```
global rbvar
rbvar = tkinter.IntVar()
rbvar.set(0)
```

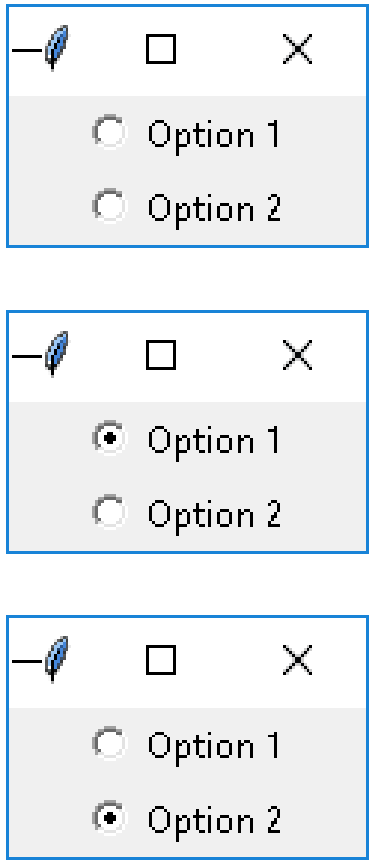
Creating a Radio Button

- When creating a radio button, we must specify
  - The window or frame it belongs to.
  - The text displayed next to the radio button.
  - The IntVar associated with the radio button.
  - The radio buttons IntVar value.

```
global rbvar
rbvar = tkinter.IntVar()
rbvar.set(0)

test_rbutton = tkinter.Radiobutton(test_window,
                                   text="Option 1",
                                   variable=rbvar
                                   value=1)
```

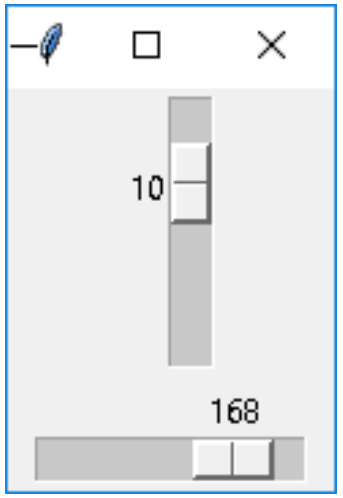
Creating a Radio Button

```
import tkinter

def main() :
     test_window = tkinter.Tk()
     test_window.wm_title("My Window")

     global rbvar
     rbvar = tkinter.IntVar()
     rbvar.set(0)

     rbutton1 = tkinter.Radiobutton(test_window,
                                    text="Option 1",
                                    variable=rbvar
                                    value=1)
     rbutton2 = tkinter.Radiobutton(test_window,
                                    text="Option 2",
                                    variable=rbvar
                                    value=2)
     rbutton1.pack()
     rbutton2.pack()
     tkinter.mainloop()

main()
```

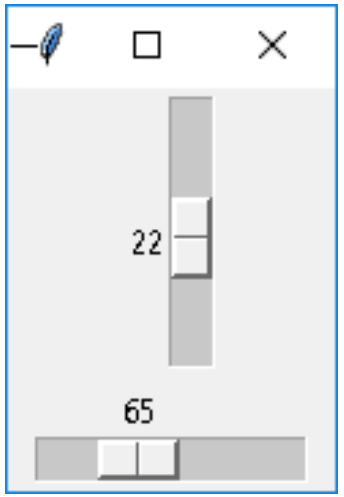


Scales

- A ***scale*** (or slider) is a rectangular component with a range of values that can be selected.
  - Includes an accompanying text label of the currently selected value.
  - Can be displayed vertically (default) or horizontally.

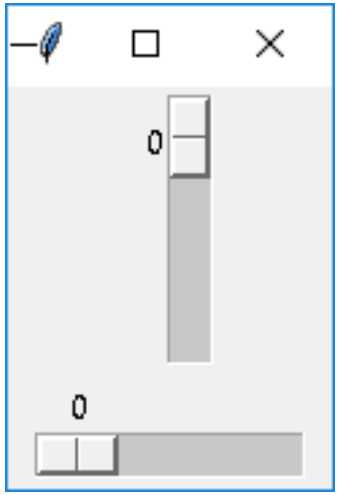


- The user clicks and drags the indicator to a desired value.



Creating a Scale

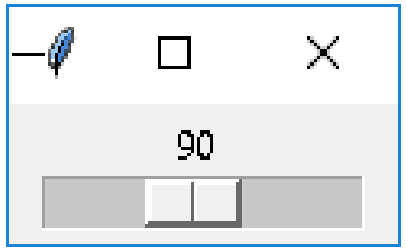
- When creating a scale, we can specify
  - The window or frame it belongs to.
  - The start of the range. (from_)
  - The end of the range. (to)
  - The orientation (for horizontal scales).




```
vscale = tkinter.Scale(test_window, from_=0, to=50)
hscale = tkinter.Scale(test_window, from_=0, to=200, orient="horizontal")
```

Creating a Scale

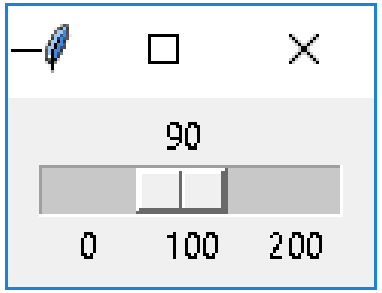
- We can set the scale's starting value with its **set** function.
  - Otherwise, the indicator will start at the first value in the range.
  - The scale's **get** function will return the currently selected (int) value.




```
hscale = tkinter.Scale(test_window, from_=0, to=200, orient="horizontal")
hscale.set(90)
```

Creating a Scale

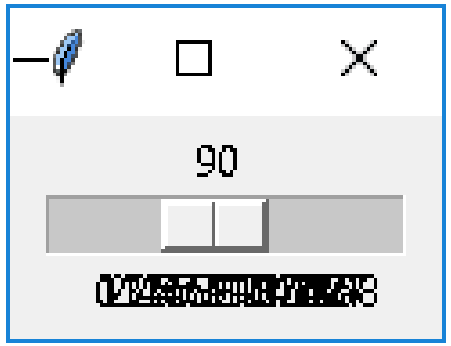
- We can display intervals along the scale using the tickinterval argument.




```
hscale = tkinter.Scale(test_window, from_=0, to=200,
                       tickinterval=100, orient="horizontal")
hscale.set(90)
```

Creating a Scale

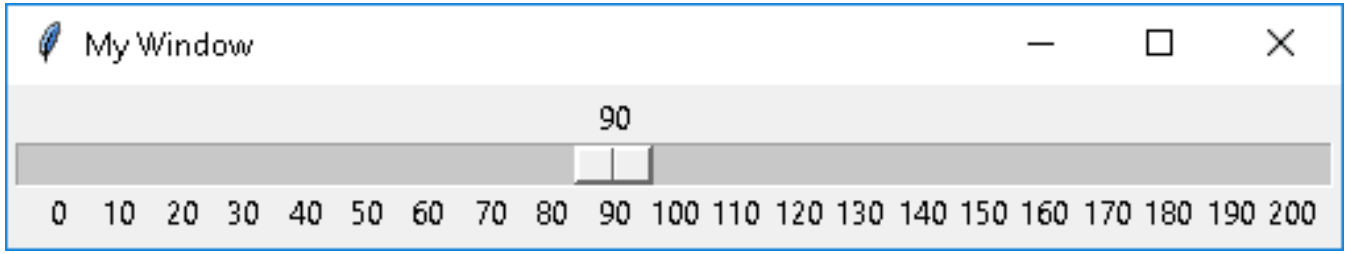
- The tick intervals may run into each other if there isn't enough room to display them all.




```
hscale = tkinter.Scale(test_window, from_=0, to=200,
                       tickinterval=10, orient="horizontal")
hscale.set(90)
```

Creating a Scale

- The scale's length argument allows us to increase the physical length of the scale.




```
hscale = tkinter.Scale(test_window, from_=0, to=200,
                       tickinterval=10, length=500, orient="horizontal")
hscale.set(90)
```