Programming for Business Computing

Graphical User Interface

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Outline

- **Basic concepts**
- Example 1A: A simple square root calculator
- Example 1B: A cool square root calculator
- Example 2: A scatter plot plotter

User interface

- Our program interact with users through a **user interface** (UI).
- User interface design is important.
  - Intuitiveness.
  - Fail-safe.
  - User experience (UX).
- So far we worked with **text-based interfaces**.
  - Command lines/consoles/terminals.
- Let’s try to build a **graphical user interface** (GUI) now.
  - Also called “front-end development”.

Developing a GUI

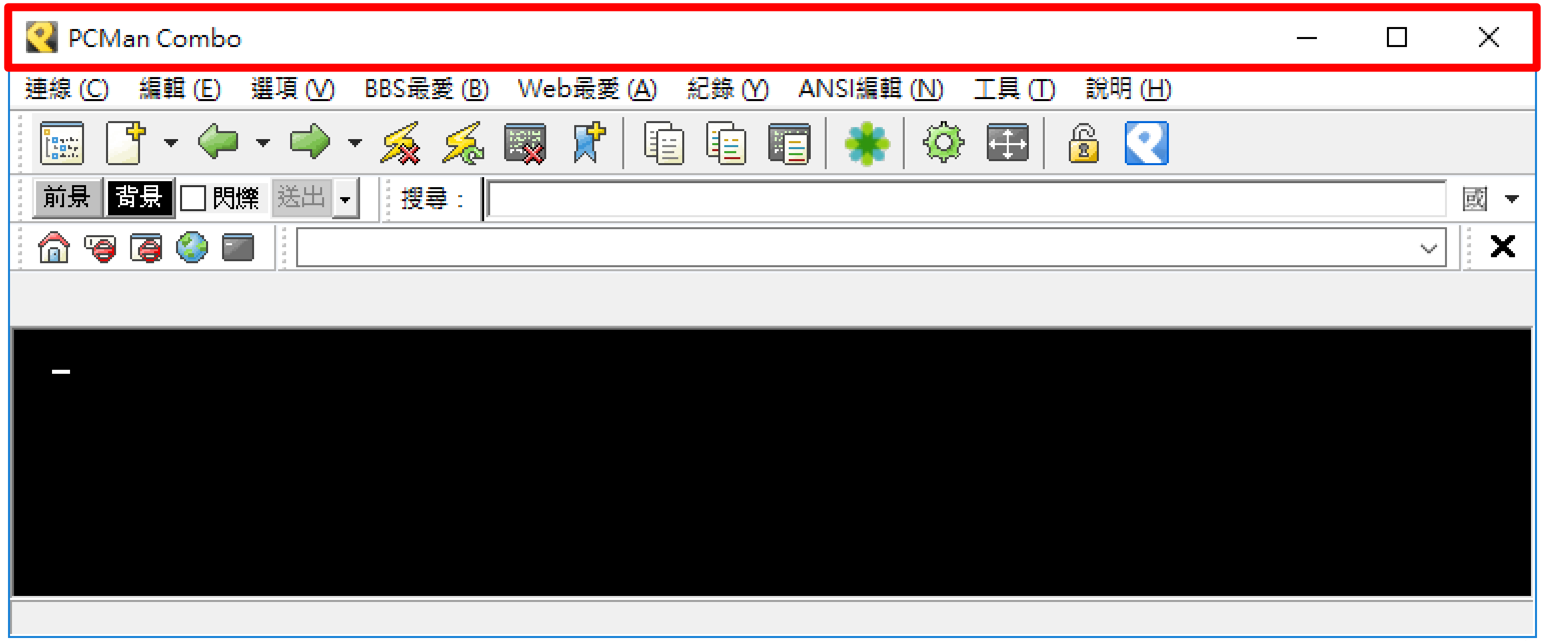
- Easier to **use** than a text-based user interface.
  - Better user experience.
- Easier to do **fail safe**.
  - Checkbox vs. entering Y/N.
  - Dropdown list vs. entering 1/2/3/4/5.
- Worse **performance**.
  - Compared to a text-based user interface.

Learning to develop a GUI

- Using Python to develop a GUI is not hard.
  - Easier than using C, C++, Java, etc.
  - However, still harder than **web development**.
- Today, you develop a GUI only if you want to make desktop software or smartphone app to sell.
  - If you just want to implement an algorithm, use a text-based UI.
  - If you want to develop an application, write a web page.
- Still, (slightly) learning how to write a GUI in Python is good.
  - Getting the fundamental ideas of GUI.
  - Getting more ideas about **classes**.
  - Getting more ideas about **software development** and **online search**.
- And getting something to demonstrate to your parents and friends.

Basic structure of a GUI: window

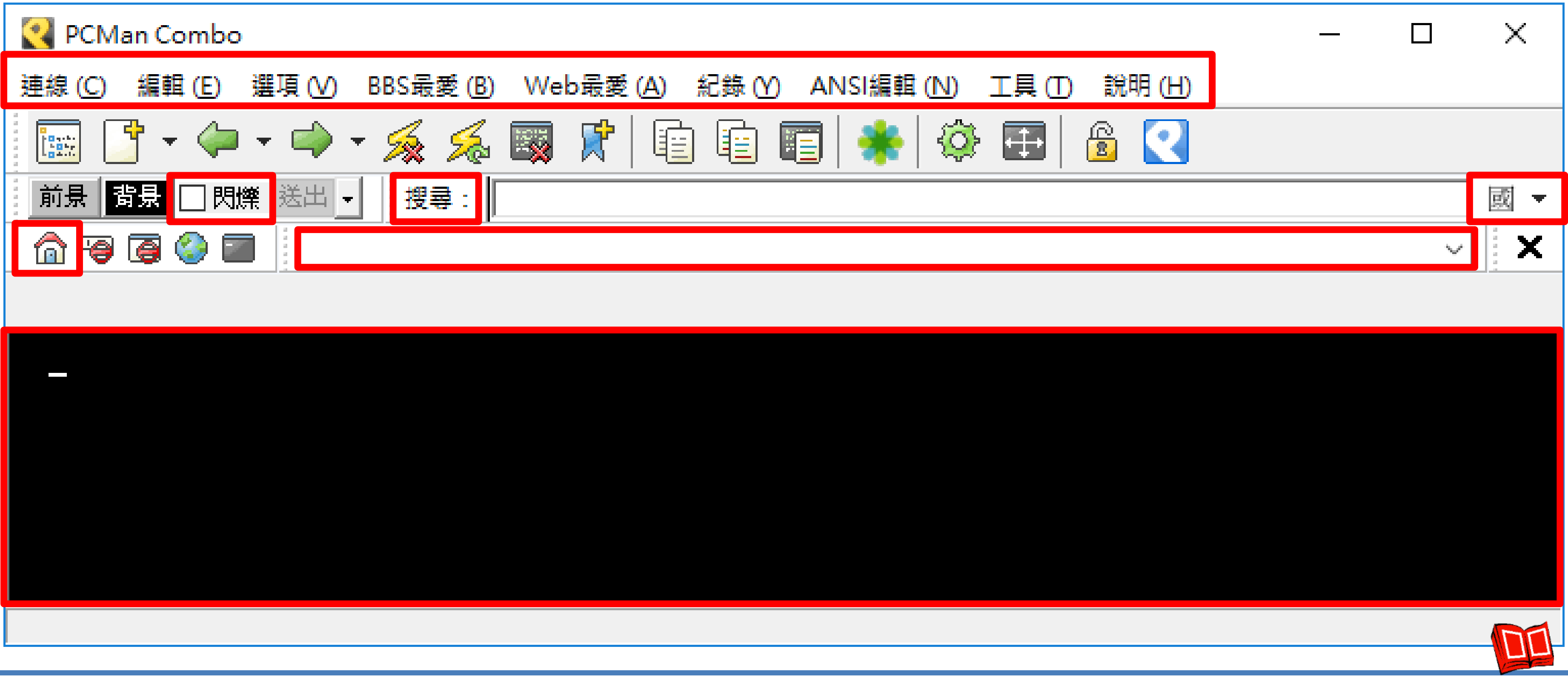
- A desktop application is typically presented in a **window** (or multiple windows).
- A window has a **header**:
  - An icon, a title, and three buttons (minimize, maximize/getting back, close).



Basic structure of a GUI: widgets

- There are **widgets** (components, elements).
  - Many of them are called icons.

buttons **checkboxes** **labels** **textboxes** **a menu** **canvases** **dropdown lists**


PCMan Combo
連線 (C) 編輯 (E) 選項 (V) BBS最愛 (B) Web最愛 (A) 紀錄 (Y) ANSI編輯 (N) 工具 (T) 說明 (H)
前景
背景
□ 閃爍
送出
搜尋：
國

Our GUI development

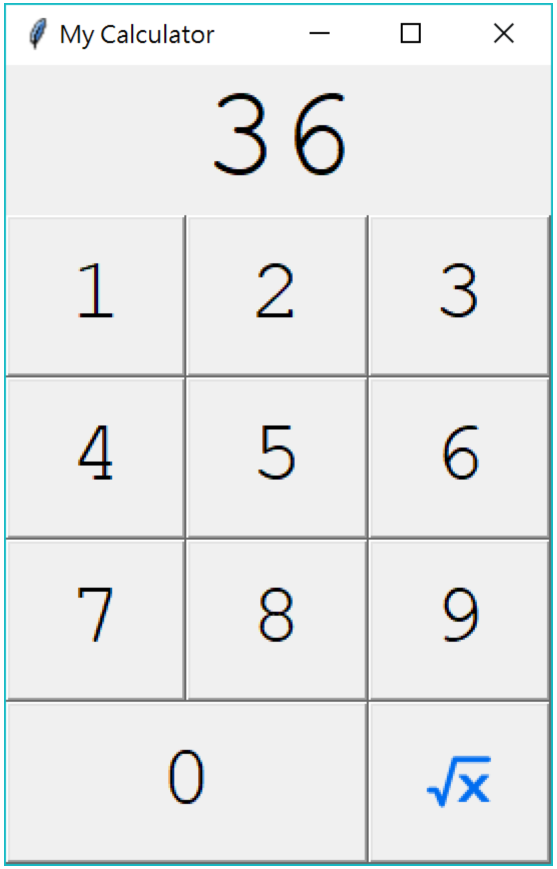
- To develop a GUI, we first create a window.
  - We will write a **class** by “inheriting” an existing window class in a library.
- We then create components by creating **objects** (using existing classes)
  - Button objects, label objects, etc.
  - They are **member variables** of our window class.
  - We specify their looks and locations by modifying their **member variables**.
- Finally, we determine their behaviors.
  - We define **member functions** of our window class.
  - We specify the function to invoke upon an **event** (e.g., when a button is clicked).
- The example programs are for Windows.
  - For Mac, please refer to the supplemental handout.

Outline

- Basic concepts
- **Example 1A: A simple square root calculator**
- Example 1B: A cool square root calculator
- Example 2: A scatter plot plotter

A square root calculator

- Our first example is a square root calculator.
  - A simpler version of a calculator.
  - A user may click on the number pad to enter a number (as a nonnegative integer).
  - She may then click on the square root icon to get the square root of the input number (as a float number rounded to the second digit after the decimal point).
- We need to:
  - Create a window.
  - Create one label and eleven buttons.
  - Implement event-triggered functions.
  - Arrange them nicely.



Calculator 0.1: Creating a window

- First, we import **tkinter** the standard Python library for creating GUI, and give it an alias **tk**.
- We then write a class **Calculator** by **inheriting** from a class **Frame**.
  - **Frame** is a class defining an **"empty" window frame**.
  - To inherit from a class, put the class name inside the pair of parentheses.
  - Inheriting an existing class allows our own class having everything defined in the "parent class".

```
import tkinter as tk

class Calculator(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()

cal = Calculator()
cal.master.title("My Calculator")
cal.mainloop()
```

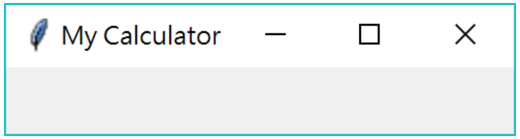
Calculator 0.1: Creating a window

- We then define our **constructor**:
  - Invoking the parent's constructor.
  - Invoking a member function (defined in **`Frame`**) to prepare "grids" to place widgets.

```
import tkinter as tk

class Calculator(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()

cal = Calculator()
cal.master.title("My Calculator")
cal.mainloop()
```

Calculator 0.1: Creating a window

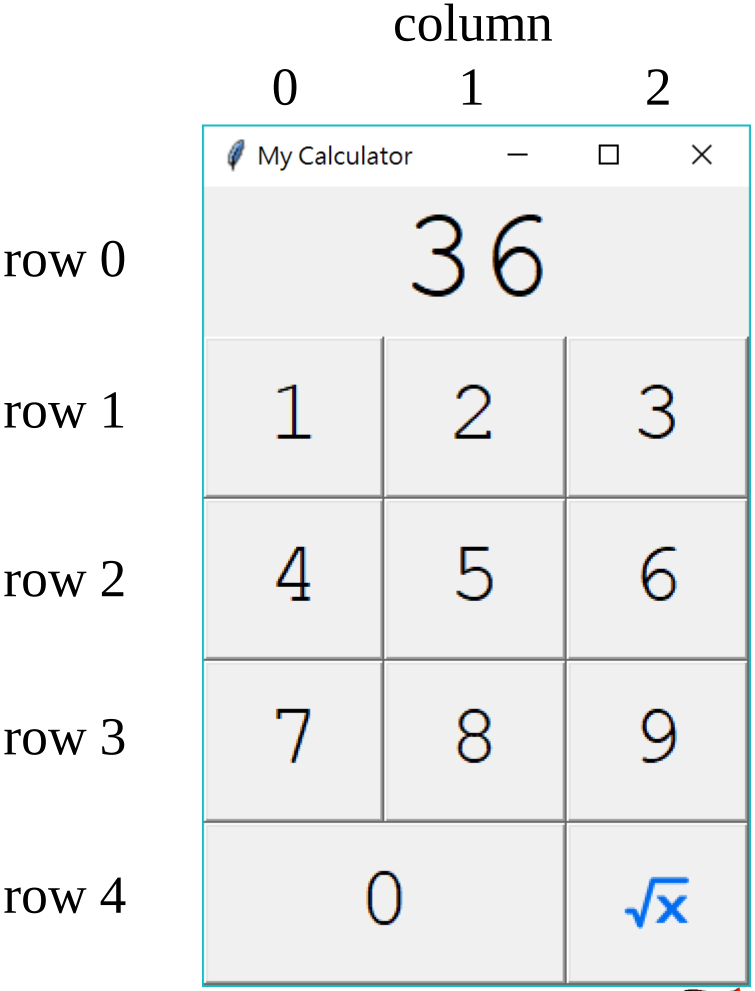
- Now we use the class to **create a `Calculator` object**.
  - First, create the object.
  - Second, use a member function (defined in **`Frame`**) to set up the title.
  - Lastly, invoke **`mainloop()`** to let it **keep listening to events** (like invoking **`input()`** and waiting for user input).
- The result:



```
import tkinter as tk

class Calculator(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()

cal = Calculator()
cal.master.title("My Calculator")
cal.mainloop()
```

Calculator 0.2: Adding widgets

- Let's add a **label** and a **button**.
- To add widgets into a window, we need to decide where to place them.
  - A window is divided into **grids**, intersections of **rows** and **columns**.
  - Here we have 5 rows and 3 columns.
  - A widget may **span** for multiple rows or columns.
- Later we will put the label at (row = 0, column = 0) and the button at (row = 1, column = 0).



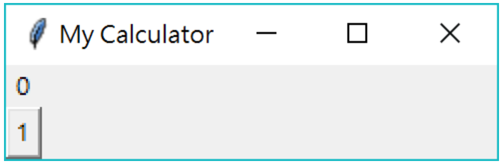
Calculator 0.2: Adding widgets

- We define a member function **`createWidgets()`**.
- We use the class **`Label`** to create a member label object.
  - The first argument says that this label **belongs to this window**.
  - The second argument sets the initial text to “0”.
  - The label object is a **member** of this window.
- The class **`Button`** works similarly.

```
import tkinter as tk

class Calculator(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()
    self.createWidgets()

  def createWidgets(self):
    self.lblNum = tk.Label(self, text = "0")
    self.btnNum1 = tk.Button(self, text = "1")
    self.lblNum.grid(row = 0, column = 0)
    self.btnNum1.grid(row = 1, column = 0)

cal = Calculator()
cal.master.title("My Calculator")
cal.mainloop()
```

Calculator 0.2: Adding widgets

- Each of the two widgets need to invoke **grid()** to set up its location.
  - We specify the **row and column indices** of each widget.
- The result:

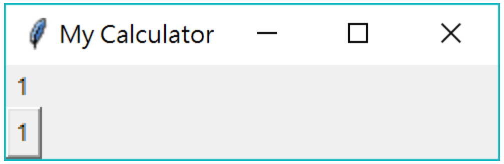


```
import tkinter as tk

class Calculator(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()
    self.createWidgets()

  def createWidgets(self):
    self.lblNum = tk.Label(self, text = "0")
    self.btnNum1 = tk.Button(self, text = "1")
    self.lblNum.grid(row = 0, column = 0)
    self.btnNum1.grid(row = 1, column = 0)

cal = Calculator()
cal.master.title("My Calculator")
cal.mainloop()
```

Calculator 0.3: Event-triggered functions

- We now implement an event-triggered function for the button.

```
def createWidgets(self):
  self.lblNum = tk.Label(self, text = "0")
  self.btnNum1 = tk.Button(self, text = "1", command = self.clickBtnNum1)
  self.lblNum.grid(row = 0, column = 0)
  self.btnNum1.grid(row = 1, column = 0)

def clickBtnNum1(self):
  self.lblNum.configure(text = "1")
```



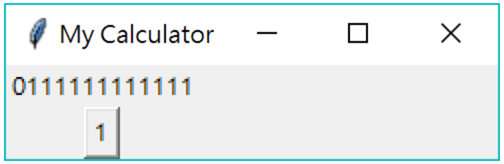
- The member function **`clickBtnNum1()`** sets the label's text to be "1".
  - By invoking the `configure()` member function.
- `command = self.clickBtnNum1` adds an **event listener** to the button.
  - When one clicks the button, a "click" event triggers **`clickBtnNum1()`**.
  - Without **`Calculator`**, this would become a (weird) global function.

Calculator 0.4: Event-triggered functions

- We now implement an event-triggered function for the button.

```
def createWidgets(self):
  self.lblNum = tk.Label(self, text = "0")
  self.btnNum1 = tk.Button(self, text = "1", command = self.clickBtnNum1)
  self.lblNum.grid(row = 0, column = 0)
  self.btnNum1.grid(row = 1, column = 0)

def clickBtnNum1(self):
  self.lblNum.configure(text = self.lblNum.cget("text") + "1")
```

- What does this implementation do?
  - `self.lblNum.cget("text")` returns the current text of a label object.
  - Clicking the button appends one more “1” to the current text.



```
0111111111111
```

Calculator 0.5: heights, widths, fonts

- Let’s adjust the look of our widgets.
- All widgets have attributes `height` and `width`.
  - For a label or button of texts, **`height`** is the number of lines and **`width`** is the number of characters.
  - For a label or button of images, **`height`** and **`width`** are pixels.
- Most widgets have the attribute `font`.
  - We may use the class `font` in **`tkinter`** to define a font object.
  - Assigning a font object to font sets the font family/type/size of the widget.
  - To import the class, add

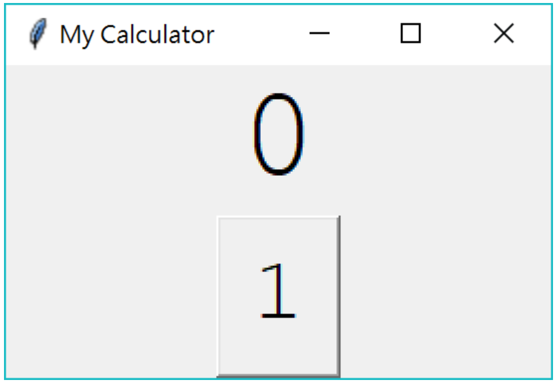
```
import tkinter.font as tkFont
```

into your Python program.

Calculator 1.0: heights, widths, fonts

```
def createWidgets(self):
  f1 = tkFont.Font(size = 48, family = "Courier New")
  f2 = tkFont.Font(size = 32, family = "Courier New")

  self.lblNum = tk.Label(self, text = "0", height = 1, width = 7, font = f1)
  self.btnNum1 = tk.Button(self, text = "1", command = self.clickBtnNum1,
                           height = 1, width = 2, font = f2)
  self.lblNum.grid(row = 0, column = 0)
  self.btnNum1.grid(row = 1, column = 0)
```

- **f1** and **f2** are two font objects.
- The label contains one line of seven 48-point Courier New characters.
- The button contains one line of two 32-point Courier New characters.



- Calculator 1.0 (which is just 0.5) is in "Calculator1.py".

Calculator 1.1: all widgets

- Let’s put all eleven buttons into the window.

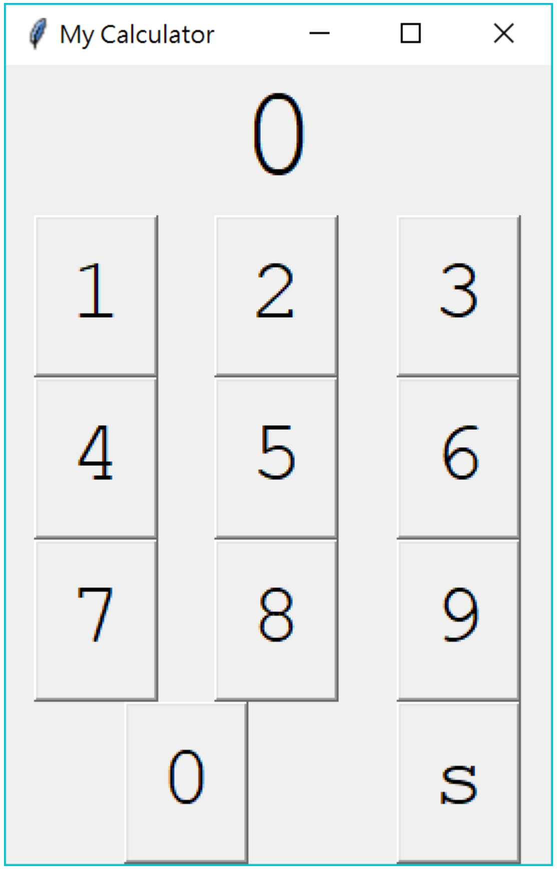
```
def createWidgets(self):
  f1 = tkFont.Font(size = 48, family = "Courier New")
  f2 = tkFont.Font(size = 32, family = "Courier New")

  self.lblNum = tk.Label(self, text = "0", height = 1, width = 7, font = f1)

  self.btnNum1 = tk.Button(self, text = "1", height = 1, width = 2,
                           command = self.clickBtnNum1, font = f2)
  self.btnNum2 = tk.Button(self, text = "2", height = 1, width = 2,
                           command = self.clickBtnNum1, font = f2)
  # let all buttons' trigger clickBtnNum1() for a while
  # btnNum3 to btnNum9 omitted
  self.btnNum0 = tk.Button(self, text = "0", height = 1, width = 2,
                           command = self.clickBtnNum1, font = f2)
  self.btnSqrt = tk.Button(self, text = "s", height = 1, width = 2,
                           command = self.clickBtnNum1, font = f2)
```

Calculator 1.1: all widgets

- Let’s set up their locations.

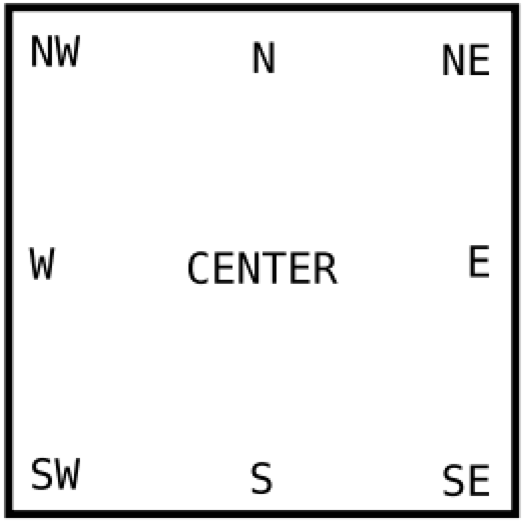
```
def createWidgets(self):
  # font and widget creation omitted

  self.lblNum.grid(row = 0, column = 0, columnspan = 3)

  self.btnNum1.grid(row = 1, column = 0)
  self.btnNum2.grid(row = 1, column = 1)
  # btnNum3 to btnNum9 omitted
  self.btnNum0.grid(row = 4, column = 0, columnspan = 2)
  self.btnSqrt.grid(row = 4, column = 2)
```

  - The attribute **columnspan** specifies the number of columns spanned by the widget.
  - The attribute **rowspan** specifies the number of columns spanned by the widget.



Calculator 1.2: expanding widgets

- How to take away the margins between widgets?
- The **`grid()`** function has a parameter **`sticky`** whose value decides how to **stick a widget** to a side (and remove the margin).
  - **`sticky = tk.E`** sticks the widget to the east (right).
  - **`sticky = tk.NE`** sticks the widget toward the north (top) and east (right).
  - To sticks the widget to multiple sides, write, e.g., **`sticky = tk.E + tk.NW`**

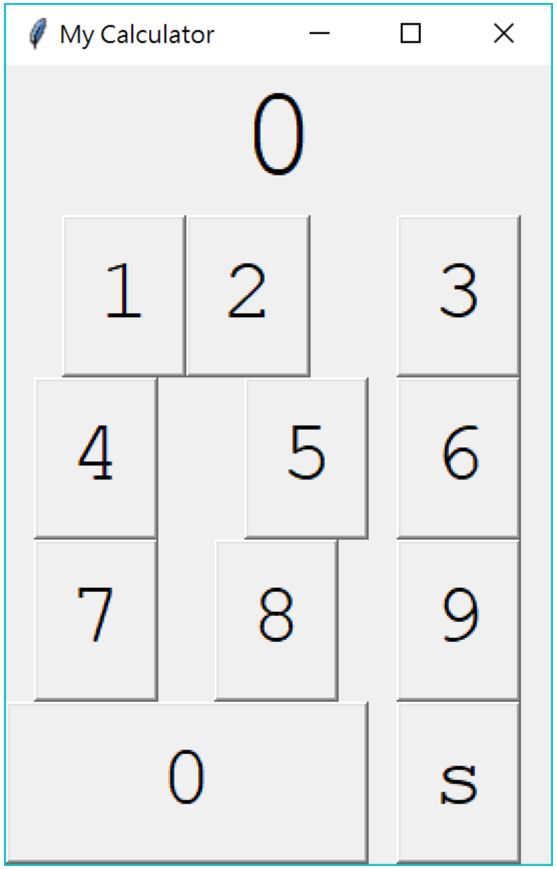


NW	N	NE
W	CENTER	E
SW	S	SE

Calculator 1.2: expanding widgets

- Let’s try it:

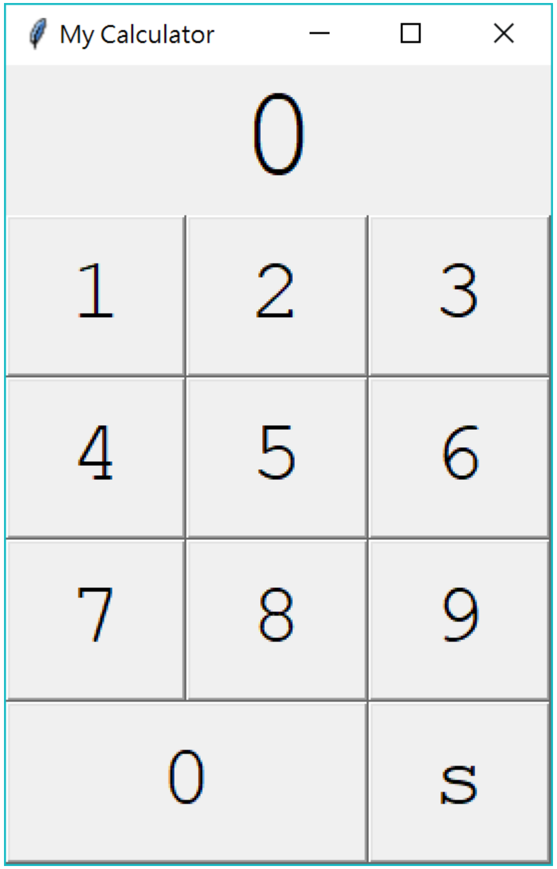
```
    self.btnNum1.grid(row = 1, column = 0, sticky = tk.E)
    self.btnNum2.grid(row = 1, column = 1, sticky = tk.W)
    self.btnNum3.grid(row = 1, column = 2, sticky = tk.N)
    self.btnNum4.grid(row = 2, column = 0, sticky = tk.S)
    self.btnNum5.grid(row = 2, column = 1, sticky = tk.NE)
    self.btnNum6.grid(row = 2, column = 2)
    self.btnNum7.grid(row = 3, column = 0)
    self.btnNum8.grid(row = 3, column = 1)
    self.btnNum9.grid(row = 3, column = 2)
    self.btnNum0.grid(row = 4, column = 0,
                      columnspan = 2,
                      sticky = tk.NE + tk.SW)
    self.btnSqrt.grid(row = 4, column = 2)
```



Calculator 1.2: expanding widgets

- Let’s remove all margins:

```
    self.lblNum.grid(row = 0, column = 0, columnspan = 3,
                     sticky = tk.NE + tk.SW)

    self.btnNum1.grid(row = 1, column = 0,
                      sticky = tk.NE + tk.SW)
    self.btnNum2.grid(row = 1, column = 1,
                      sticky = tk.NE + tk.SW)
    # btnNum3 to btnNum9 omitted
    self.btnNum0.grid(row = 4, column = 0, columnspan = 2,
                      sticky = tk.NE + tk.SW)
    self.btnSqrt.grid(row = 4, column = 2,
                      sticky = tk.NE + tk.SW)
```



Calculator 1.3: adding functions

- Let’s add a function for button 2:

```
def createWidgets(self):
    # all others omitted
    self.btnNum1 = tk.Button(self, text = "1", height = 1, width = 2,
                             command = self.clickBtnNum1, font = f2)
    self.btnNum2 = tk.Button(self, text = "2", height = 1, width = 2,
                             command = self.clickBtnNum2, font = f2)

  def clickBtnNum1(self):
    self.lblNum.configure(text = self.lblNum.cget("text") + "1")

  def clickBtnNum2(self):
    self.lblNum.configure(text = self.lblNum.cget("text") + "2")
```

- And then repeat this for all buttons (except square root).

Calculator 1.3: adding functions

- Let’s add a function for the square root button:

```
def createWidgets(self):
    # all others omitted
    self.btnSqrt = tk.Button(self, text = "s", height = 1, width = 2,
                              command = self.clickBtnSqrt, font = f2)

  def clickBtnSqrt(self):
    curNum = float(self.lblNum.cget("text"))
    self.lblNum.configure(text = str(round(math.sqrt(curNum), 2)))
```

  – Take the current number, cast it to a float, find its square root, round it, convert it to a string, and then override the current number.
- This is good, but…
  – What happens if we then click a button of any number?

Calculator 1.3: adding functions

- Let’s add a **flag** for whether we should reset the number:

```
class Calculator(tk.Frame): # all others omitted

  shouldReset = True # the flag

  def clickBtnNum1(self):
    if self.shouldReset == True:
      self.lblNum.configure(text = "1")
      self.shouldReset = False
    else:
      self.lblNum.configure(text = self.lblNum.cget("text") + "1")

  def clickBtnSqrt(self):
    curNum = float(self.lblNum.cget("text"))
    self.lblNum.configure(text = str(round(math.sqrt(curNum), 2)))
    self.shouldReset = True
```

- Should we modify the functions for other buttons in the same way?

Calculator 2.0: adding functions

- Let’s use a more modularized way:

```
class Calculator(tk.Frame): # all others omitted

  def setNumStr(self, content):
    if self.shouldReset == True:
      self.lblNum.configure(text = content)
      self.shouldReset = False
    else:
      self.lblNum.configure(text = self.lblNum.cget("text") + content)

  def clickBtnNum1(self):
    self.setNumStr("1")

  def clickBtnNum2(self):
    self.setNumStr("2")
```

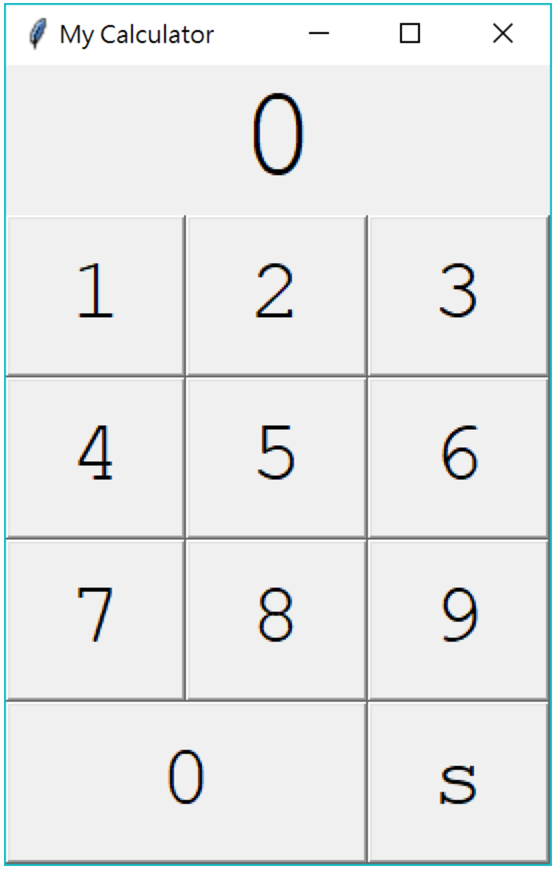
- Calculator 2.0 (which is just 1.3) is in “Calculator2.py”

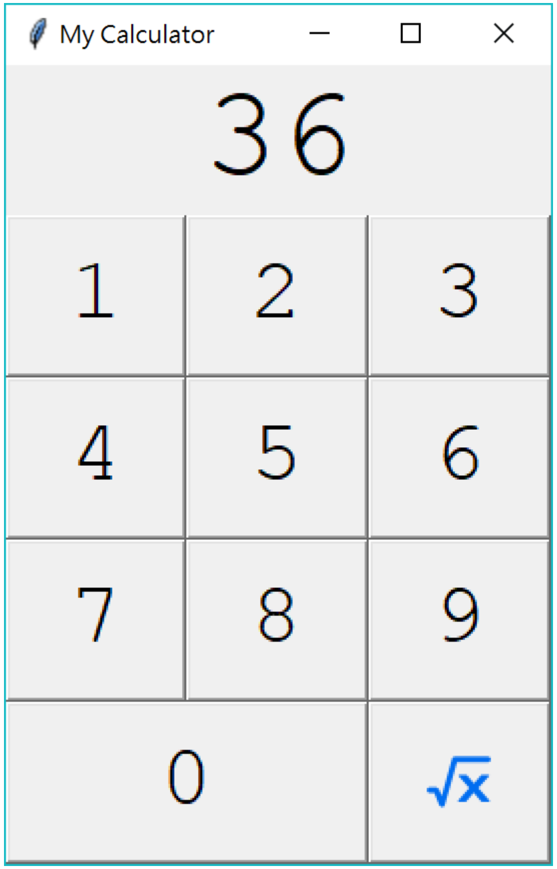
Outline

- Basic concepts
- Example 1A: A simple square root calculator
- **Example 1B: A cool square root calculator**
- Example 2: A scatter plot plotter

Calculator 2.1: the square root image

- The current version is good, but the label of the square root button is not good.
- Which one do you prefer?
- Let’s use an image rather than a text as the label.





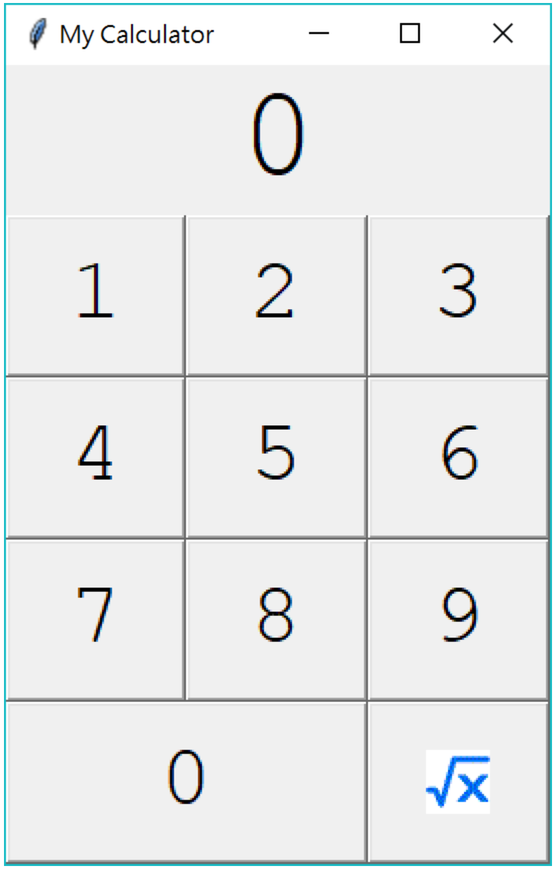
Calculator 2.1: the square root image

- First, we need to prepare an image of square root.

- While many images are in PNG, JPG, and BMP format, the default **tkinter** class **PhotoImage** only support GIF, PGM, PPM, and XBM formats.
- Suppose that we have a GIF image, we do:

```
    self.imageSqrt = tk.PhotoImage(file = "sqrt.gif")
    self.btnSqrt = tk.Button(self, image = self.imageSqrt,
                             command = self.clickBtnSqrt)
```

  - Don’t forget to use cmd to run the program.
- Okay but not perfect.

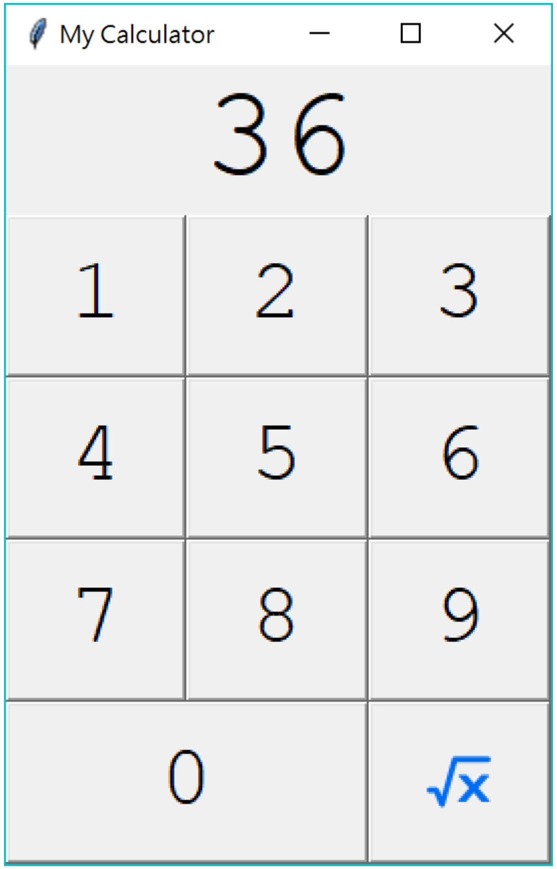


Calculator 3.0: using PIL

- To use a PNG format, we may install the library **PIL** (Python Image Library) by install **Pillow**.
  - https://python-pillow.org/.
  - To install Pillow, run "`pip install Pillow`" in cmd.
- We now write:

```
from PIL import ImageTk

class Calculator(tk.Frame):
    # all others omitted
    self.imageSqrt = ImageTk.PhotoImage(file = "sqrt.png")
    self.btnSqrt = tk.Button(self, image = self.imageSqrt,
                             command = self.clickBtnSqrt)
```

- Calculator 3.0 (which is just 2.2) is in "Calculator3.py".



Challenge

- If we write

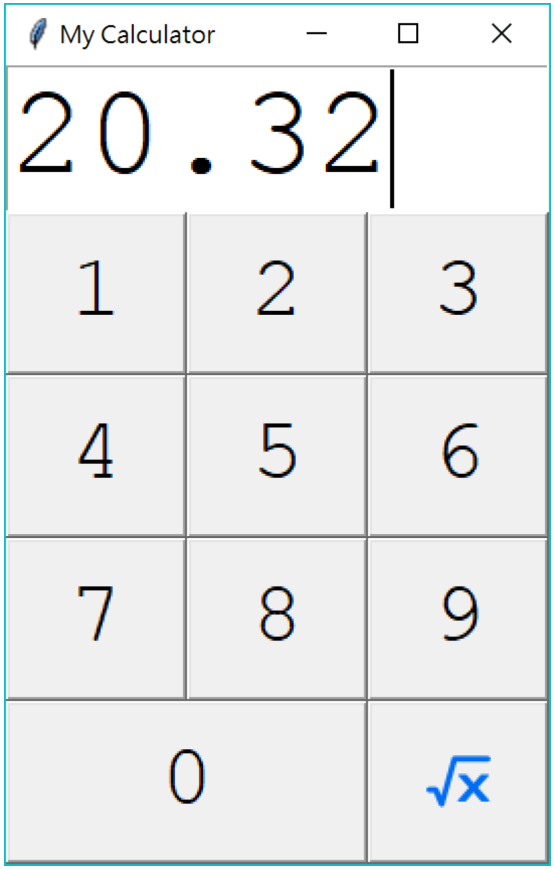
```
from PIL import ImageTk

class Calculator(tk.Frame):
    # all others omitted
    imageSqrt = Image.PhotoImage(file = "sqrt.png")
    self.btnSqrt = tk.Button(self, image = imageSqrt,
                             command = self.clickBtnSqrt)
```

the calculator works well, but the image disappears!

- Why?

Calculator 4.0: textbox

- Let’s allow a user to type in numbers.



Calculator 4.0: textbox

- First, we change the label to a textbox, i.e., we change

```
self.lblNum = tk.Label(self, height = 1, width = 7, text = "0", font = f1)
```

to

```
self.txtNum = tk.Text(self, height = 1, width = 7, font = f1)
```

- We also change the code of setting its location, i.e., we change

```
self.lblNum.grid(row = 0, column = 0, columnspan = 3, sticky = tk.NE + tk.SW)
```

to

```
self.txtNum.grid(row = 0, column = 0, columnspan = 3, sticky = tk.NE + tk.SW)
```

Calculator 4.0: textbox

- When one clicks a number button, we change

```
def setNumStr(self, content):
  if self.shouldReset == True:
    self.lblNum.configure(text = content)
    self.shouldReset = False
  else:
    self.lblNum.configure(text = self.lblNum.cget("text") + content)
```

to

```
def setNumStr(self, content):
   if self.shouldReset == True:
     self.txtNum.delete("1.0", tk.END)    # 1.0: the first line,
     self.txtNum.insert("1.0", content)   #      the 0th character
     self.shouldReset = False             # tk.END: the last character

   else:
     self.txtNum.insert(tk.END, content)
```

- When one types a number, the textbox always gets that number inserted.

Calculator 4.0: textbox

- When one clicks the square root button, we change

```
def clickBtnSqrt(self):
    curNum = float(self.lblNum.cget("text"))
    self.lblNum.configure(text = str(round(math.sqrt(curNum), 2)))
    self.shouldReset = True
```

to

```
def clickBtnSqrt(self):
    curNum = float(self.txtNum.get("1.0", tk.END))
    self.txtNum.delete("1.0", tk.END)
    self.txtNum.insert("1.0", str(round(math.sqrt(curNum), 2)))
    self.shouldReset = True
```

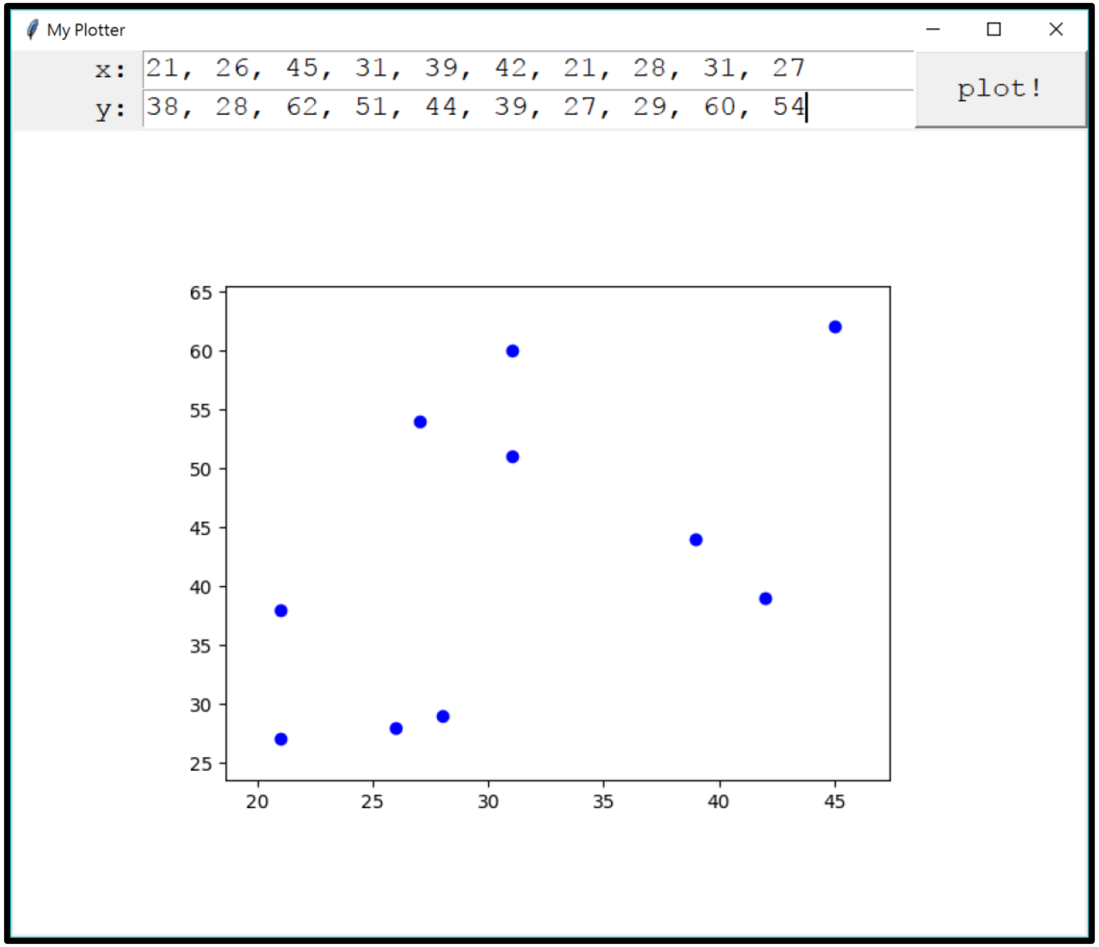
- Calculator 4.0 is in “Calculator4.py”.

Outline

- Basic concepts
- Example 1A: A simple square root calculator
- Example 1B: A cool square root calculator
- **Example 2: A scatter plot plotter**

A scatter plot plotter

- In our second example, we will:
  - Use two textboxes to let users input comma-separated values.
  - Use a canvas to place a scatter plot based on the user input.



x: 21, 26, 45, 31, 39, 42, 21, 28, 31, 27
y: 38, 28, 62, 51, 44, 39, 27, 29, 60, 54

Plotter 1.0: window and widgets

```
import tkinter as tk
import tkinter.font as tkFont

class Plotter(tk.Frame):

  def __init__(self):
    tk.Frame.__init__(self)
    self.grid()
    self.createWidgets()

  def createWidgets(self):
    f = tkFont.Font(size = 16, family = "Courier New")
    self.lblX = tk.Label(self, text = "x:", height = 1, width = 3, font = f)
    self.lblY = tk.Label(self, text = "y:", height = 1, width = 3, font = f)
    self.txtX = tk.Text(self, height = 1, width = 40, font = f)
    self.txtY = tk.Text(self, height = 1, width = 40, font = f)
    self.btnLoad = tk.Button(self, text = "plot!", height = 1, width = 5, font = f)
    self.cvsMain = tk.Canvas(self, width = 800, height = 600, bg = "white")
```

Plotter 1.0: window and widgets

```
    self.lblX.grid(row = 0, column = 0, sticky = tk.E)
    self.lblY.grid(row = 1, column = 0, sticky = tk.E)
    self.txtX.grid(row = 0, column = 1, sticky = tk.NE + tk.SW)
    self.txtY.grid(row = 1, column = 1, sticky = tk.NE + tk.SW)
    self.btnLoad.grid(row = 0, rowspan = 2, column = 2, sticky = tk.NE + tk.SW)
    self.cvsMain.grid(row = 2, column = 0, columnspan = 3, sticky = tk.NE + tk.SW)

pl = Plotter()
pl.master.title("My Plotter")
pl.mainloop()
```

- The sticky setting pushes the texts in the two labels to the right.
- Plotter 1.0 is in “Plotter1.py”.

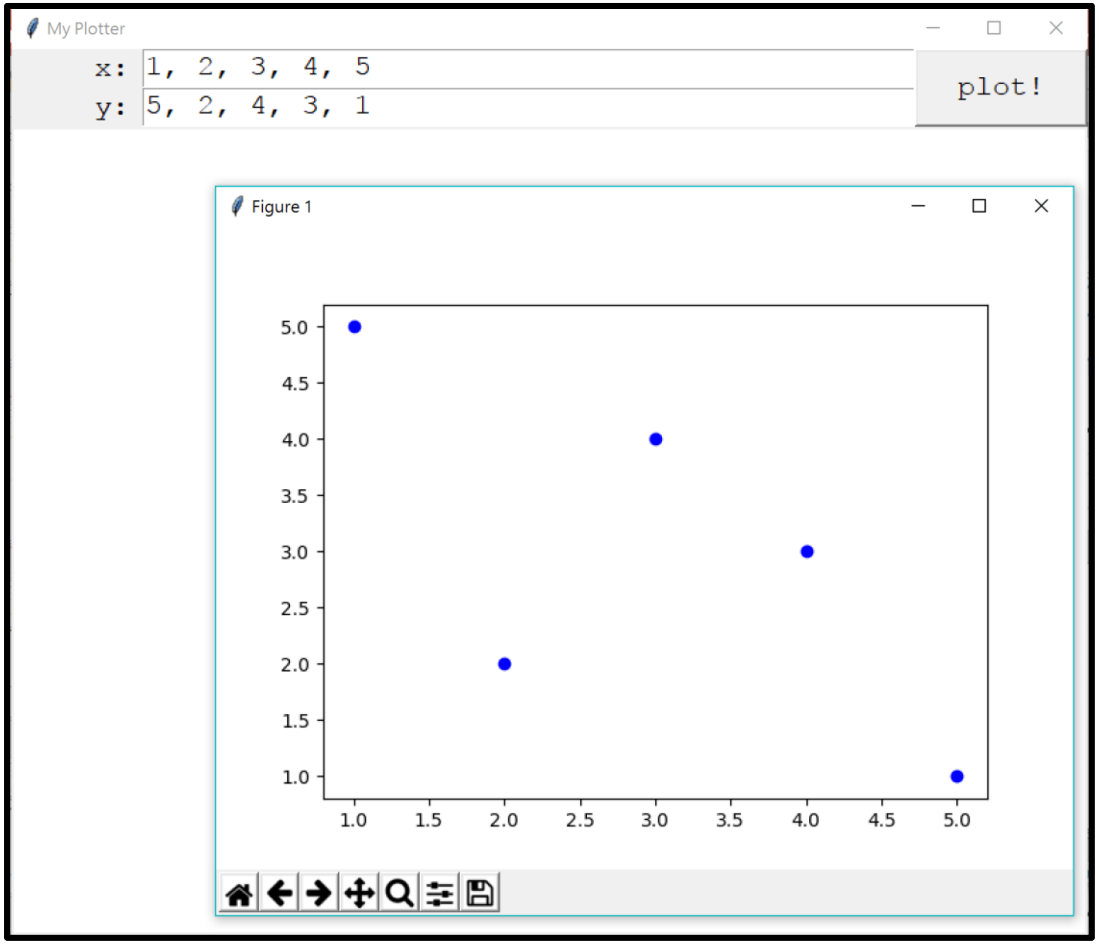
Plotter 2.0: drawing scatter plots

- To draw a scatter plot, we first:
  - Extract the texts in the two textboxes (and process them).
  - Draw a scatter plot by **`matplotlib.pyplot`**.

```
import matplotlib.pyplot as pyplot

class Plotter(tk.Frame): # all others omitted
  def createWidgets(self):
    self.btnLoad = tk.Button(self, text = "plot!", height = 1, width = 5,
                             command = self.clickBtnLoad, font = f)

 def clickBtnLoad(self):
    x = self.txtX.get("1.0", tk.END).split(",") # 1.0: the first line,
    for i in range(len(x)):                     #      the 0th character
      x[i] = float(x[i])                        # tk.END: until the last character
    y = self.txtY.get("1.0", tk.END).split(",")
    for i in range(len(y)):
      y[i] = float(y[i])
    pyplot.plot(x, y, 'bo')
    pyplot.show()
```

Plotter 2.0: drawing scatter plots

- That was good, but:
  - The scatter plot is not on the canvas.
  - **`xlim`** and **`ylim`** of the scatter plot is not set properly.
- Plotter 2.0 is in “Plotter2.py”.



My Plotter

x: 1, 2, 3, 4, 5

y: 5, 2, 4, 3, 1

Plotter 3.0: revision

- Let’s write a function for making a “nice” scatter plot.
  - And **save it as a file**.

```
class Plotter(tk.Frame): # all others omitted

  def makeScatter(self, x, y):
    pyplot.figure()            # to create a new figure
    pyplot.plot(x, y, 'bo')

    rangeX = max(x) - min(x)
    pyplot.xlim(min(x) - rangeX * 0.1, max(x) + rangeX * 0.1)
    rangeY = max(y) - min(y)
    pyplot.ylim(min(y) - rangeY * 0.1, max(y) + rangeY * 0.1)

    pyplot.savefig("temp.png")
```

Plotter 3.0: revision

- Now we put the saved file onto the canvas.

```
from PIL import ImageTk

class Plotter(tk.Frame): # all others omitted

  def clickBtnLoad(self):
    x = self.txtX.get("1.0", tk.END).split(",")
    for i in range(len(x)):
      x[i] = float(x[i])
    y = self.txtY.get("1.0", tk.END).split(",")
    for i in range(len(y)):
      y[i] = float(y[i])

    self.makeScatter(x, y)

    self.imageMain = ImageTk.PhotoImage(file = "temp.png")
    self.cvsMain.create_image(400, 300, image = self.imageMain, anchor = tk.CENTER)
```

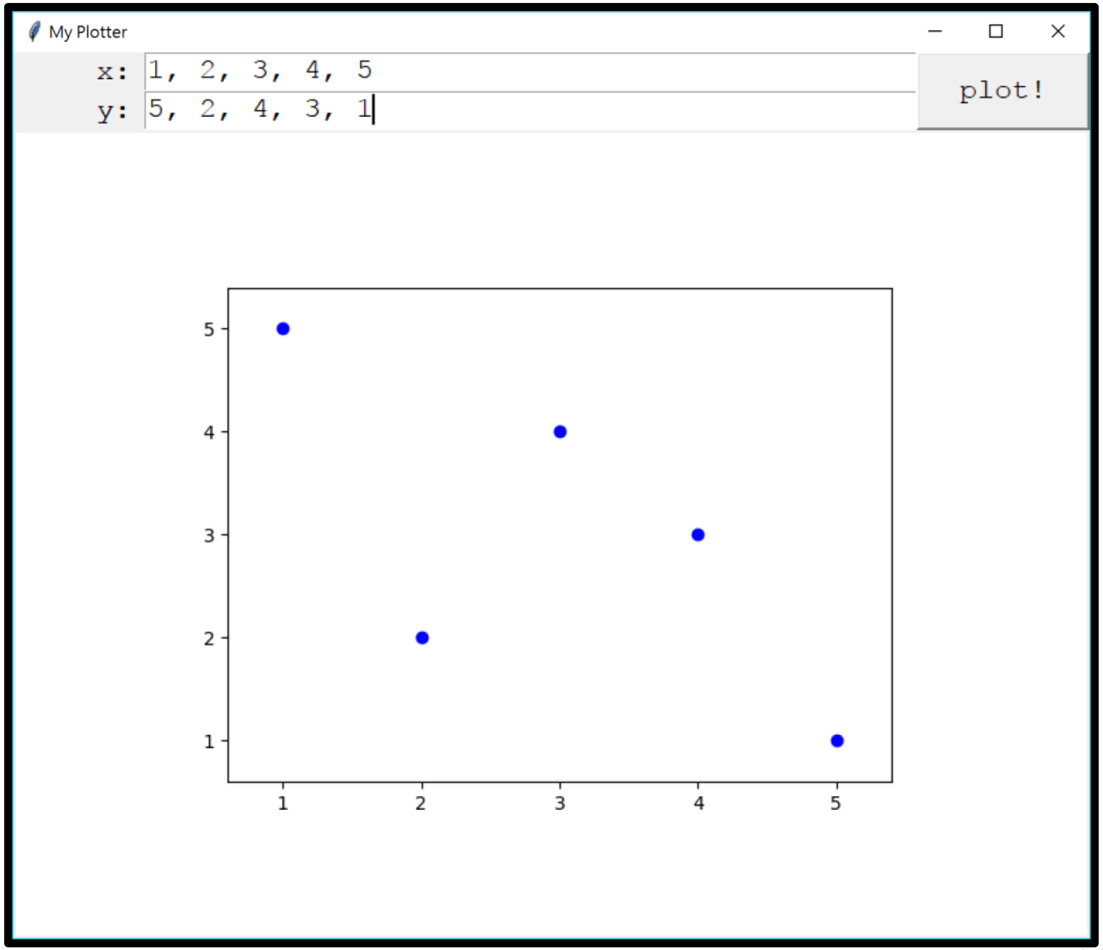
Plotter 3.0: revision

- Let’s delete the file after it is used.

```
import os

class Plotter(tk.Frame): # all others omitted

  def clickBtnLoad(self):
    # string processing...

    self.makeScatter(x, y)

    self.imageMain = ImageTk.PhotoImage(file = "temp.png")
    self.cvsMain.create_image(400, 300, image = self.imageMain, anchor = tk.CENTER)
    os.system("del temp.png")
```

Plotter 3.0: revision

- Plotter 3.0 is in “Plotter3.py”.



x: 1, 2, 3, 4, 5

y: 5, 2, 4, 3, 1

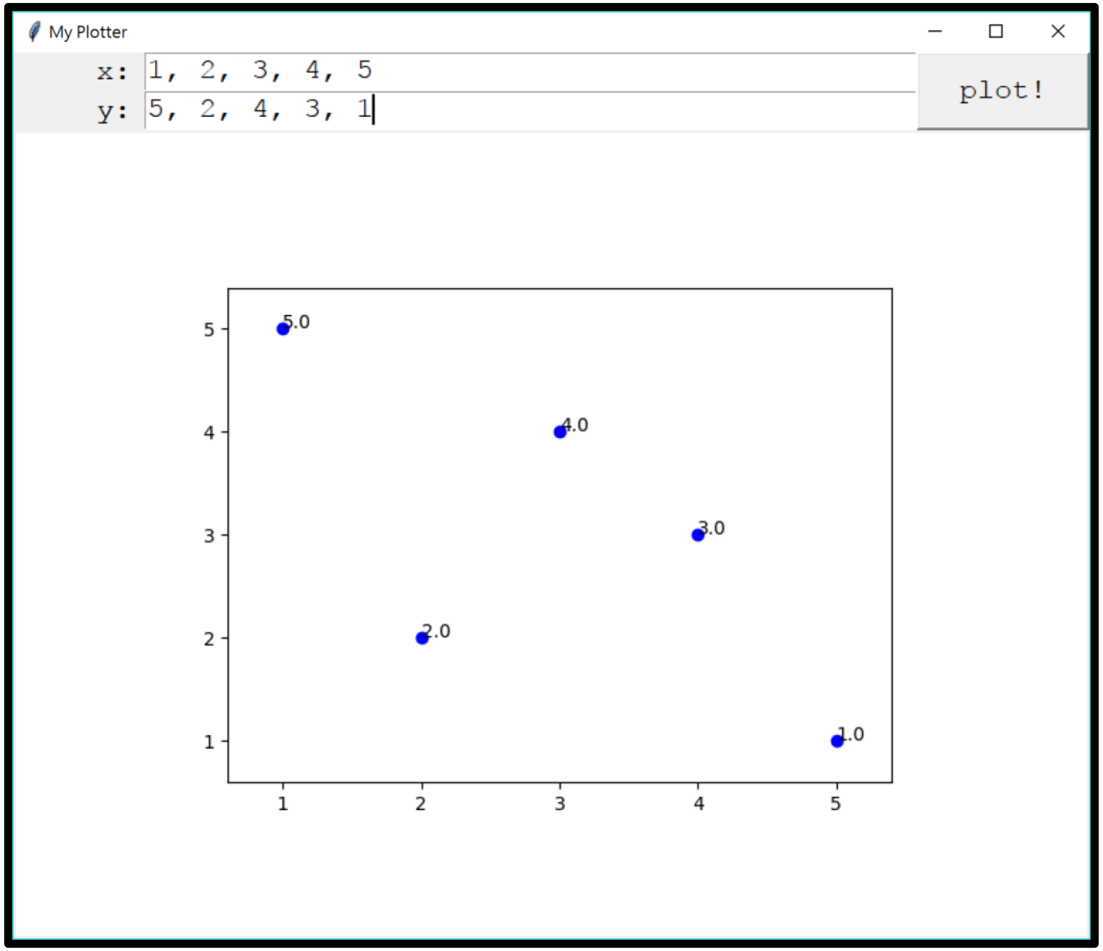
Plotter 4.0: coordinates

- May we add coordinate labels onto the plot? Yes!

```
class Plotter(tk.Frame): # all others omitted

  def makeScatter(self, x, y):
    fig = pyplot.figure()
    ax = fig.add_subplot(111)

    rangeX = max(x) - min(x)
    ax.set_xlim(min(x) - rangeX * 0.1, max(x) + rangeX * 0.1)
    rangeY = max(y) - min(y)
    ax.set_ylim(min(y) - rangeY * 0.1, max(y) + rangeY * 0.1)

    pyplot.plot(x, y, 'bo')

    for i, j in zip(x, y):
      ax.annotate(str(j), xy = (i, j))

    pyplot.savefig("temp.png")
```

Plotter 4.0: coordinates

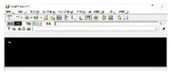
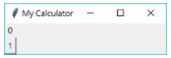
- Plotter 4.0 is in “Plotter4.py”.



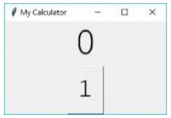
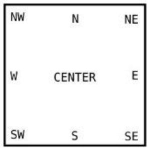
Remarks

- GUI development motivated OOP.
- Most of us in the future will not write programs to build a GUI.
- However, the following concepts are good to know:
  - Objects, classes, and inheritance.
  - Modularization (e.g., each button is an object).
  - Event listeners.
- To add more widgets and make more powerful applications, search online!

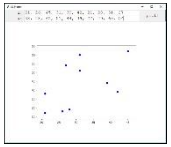
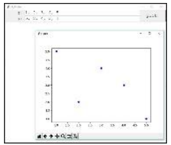
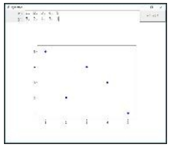
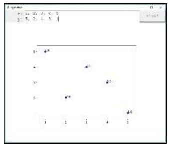
版權聲明

序	頁	作品	版權標章	作者 / 來源
1	1-53		cc BY NC ND	台灣大學 孔令傑, CC BY-NC-ND 3.0
2	6 7		📖	2005–2017 PCMan BBS Project, 洪任諭 http://pcman.ptt.cc/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
3	10 14 31 33		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
4	13		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
5	16		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
6	17		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited

版權聲明

序	頁	作品	版權標章	作者 / 來源
7	18	 My Calculator 0111111111111 1	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
8	19	 My Calculator 0 1	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
9	22	 0 1 2 3 4 5 6 7 8 9 0 s	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
10	23	 NW N NE W CENTER E SW S SE	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
11	24	 0 1 2 3 4 5 6 7 8 9 0 s	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
12	25 31	 0 1 2 3 4 5 6 7 8 9 0 s	📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited

版權聲明

序	頁	作品	版權標章	作者 / 來源
13	32		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
14	35		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
15	40		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
16	44		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
17	48		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited
18	50		📖	Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/10/13 visited