Python GUI Programming

Using Tkinter

Acknowledgment: these slides are based on material by: Velian Pandeliev, Diane Horton, Michael Samozi, Jennifer Campbell, and Paul Gries from CS UoT

Creating Buttons

- A button widget is created by using the Button constructor. For example,

```
button1 = Button(root, text = “Button 1”)
```

- The position of a widget within the root widget may be specified by passing an argument to pack():

```
button1.pack(side = LEFT)
```

- button1 will be placed as far to the left within its parent as possible.
- Other possible values for side are RIGHT, TOP, and BOTTOM

button1.py

```
from Tkinter import *

root = Tk()
root.title("Buttons")
button1 = Button(root, text = "Button 1")
button1.pack(side = LEFT)
button2 = Button(root, text = "Button 2")
button2.pack(side = LEFT)
button3 = Button(root, text = "Button 3")
button3.pack(side = LEFT)
root.mainloop()
```

button1.py

- When the user clicks a button in a GUI app, an event is generated.
- The buttons in button1.py look very nice, but they don't do anything interesting when clicked!
- We can remedy this problem by <u>specifying a command (a function) to be executed</u> whenever the button is pressed. This is done via the command parameter of the Button constructor:

```
button1 = Button(root, “Button1”, command = b1)
```

button2.py, part 1

```
from Tkinter import *

def b1():
    print "Button 1 was pressed"

def b2():
    print "Button 2 was pressed"

def b3():
    print "Button 3 was pressed"
```

button2.py, part 2

```
root = Tk()
root.title("Buttons")
button1 = Button(root, text = "Button 1",\
                       command = b1)
button1.pack(side = LEFT)
button2 = Button(root, text = "Button 2",\
                       command = b2)
button2.pack(side = LEFT)
button3 = Button(root, text = "Button 3",\
                       command = b3)
button3.pack(side = LEFT)
root.mainloop()
```

Button Size

- A button’s size can be specified with the **height** and **width** parameters.
- Example:

```
but1 = Button(root, text=“Button 1”, \
                   height = 2, width = 25)
```

  - Since the button displays text, height and width are in text units.
  - If a button displays an image, height and width are in pixel units

Button Color

- The background and foreground colors of a widget are specified via the **background** (or **bg**) and **foreground** (or **fg**) options.
- Colors can be expressed using
  - the format “#RRGGBB”, where RR, GG, and BB are hexadecimal digits.
  - predefined colors such as “red” or “blue”
- Example:

```
rgb_bg = “#c080c8”
but1 = Button(root, bg = rgb_bg)
```

button3.py (1)

```
from Tkinter import *

# background colors for buttons
tk_bg1 = "#80c0c8"
tk_bg2 = "#c8c080"
tk_bg3 = "#c080c8"

# definitions for b1, b2, and b3 are the
# same as those in button2.py
```

button3.py (2)

```
root = Tk()
root.title("Buttons")

button1 = Button(root, text = "Button 1",\
                 height = 2, width = 25, \
                 bg = tk_bg1, command=b1)
button1.pack(side = TOP)
```

button3.py (3)

```
button2 = Button(root, text = "Button 2",\
                 height = 2, width = 25, \
                 bg = tk_bg2, command = b2)
button2.pack(side = TOP)

button3 = Button(root, text = "Button 3",\
                 height = 2, width = 25, \
                 bg = tk_bg3, command = b3)
button3.pack(side = TOP)

root.mainloop()
```

Text entry widgets

- The Entry widget allows a user to enter a single line of text.
- An Entry is defined as follows:

```
text1 = Entry(parent)
text2 = Entry(parent, width = 20)
```

- The get() method allows the program to fetch the string contents of an Entry, as in

```
contents = text1.get()
```

Frame widgets

- A Frame is a widget whose purpose is to hold other widgets
- To declare a Frame and place it in its parent widget:

```
frm = Frame(parent)
frm.pack(side = TOP)
```

- To add widgets to the frame:

```
lab1 = Label(frm, text = “Name:”)
lab1.pack(side = LEFT)
text1 = Entry(frm, width = 20)
text1.pack(side = RIGHT)
```

textentry1.py (1)

```
from Tkinter import *

def getName():
    print text1.get()

root = Tk()
root.title("Buttons")

frame = Frame(root)
frame.pack(side = TOP)
```

textentry1.py (2)

```
# put a label and entry in the frame:

label1 = Label(frame, \
               text = "Enter your name:")
label1.pack(side = LEFT)

text1 = Entry(frame, width = 20)
text1.pack(side = LEFT)
```

textentry1.py (3)

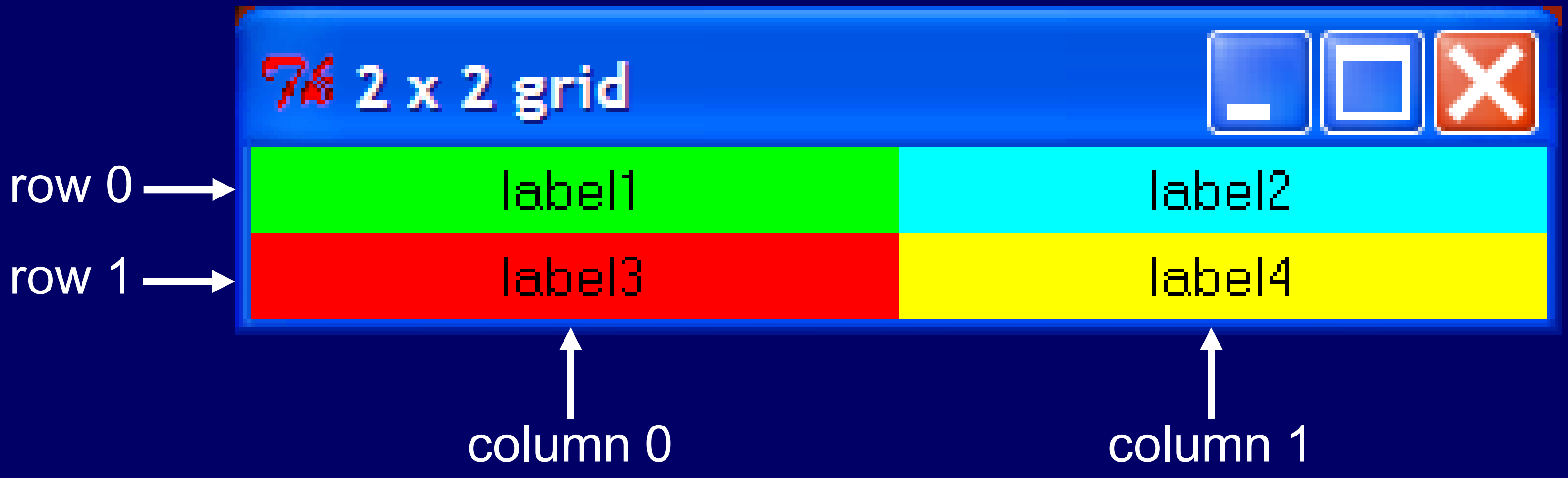
```
# put a button at the bottom of the main
# window

button1 = Button(root, text = "Accept",\
                 width = 30, \
                 bg = "#80c0c8", \
                 command = getName)
button1.pack(side = BOTTOM)

root.mainloop()
```

The grid geometry manager

- We have used the **pack** geometry manager to organize widgets in a window
- The **grid** geometry manager is used to place widgets in a rectangular grid
- The grid() method is used to determine a widget's position in a grid. Useful options include:
  - **`row`**—numbers begin at 0
  - **`column`**—numbers begin at 0
  - **`columnspan`**—allows widget to occupy several columns
  - **`sticky`**—determines position in column

A simple 2 x 2 grid

- We will create a simple but useless program to illustrate the grid manager.
- This program displays a 2 x 2 grid of labels:



twobytwo.py (1)

```
from Tkinter import *

root = Tk()
root.title("2 x 2 grid")
frame = Frame(root)
frame.pack(side = TOP)

label1 = Label(frame, \
               text = "label1", \
               width = 20, \
               bg = "green")
label1.grid(row = 0, column = 0)
```

twobytwo.py (2)

```
label2 = Label(frame, \
               text = "label2", \
               width = 20, \
               bg = "cyan")
label2.grid(row = 0, column = 1)

label3 = Label(frame, \
               text = "label3", \
               width = 20, \
               bg = "red")
label3.grid(row = 1, column = 0)
```

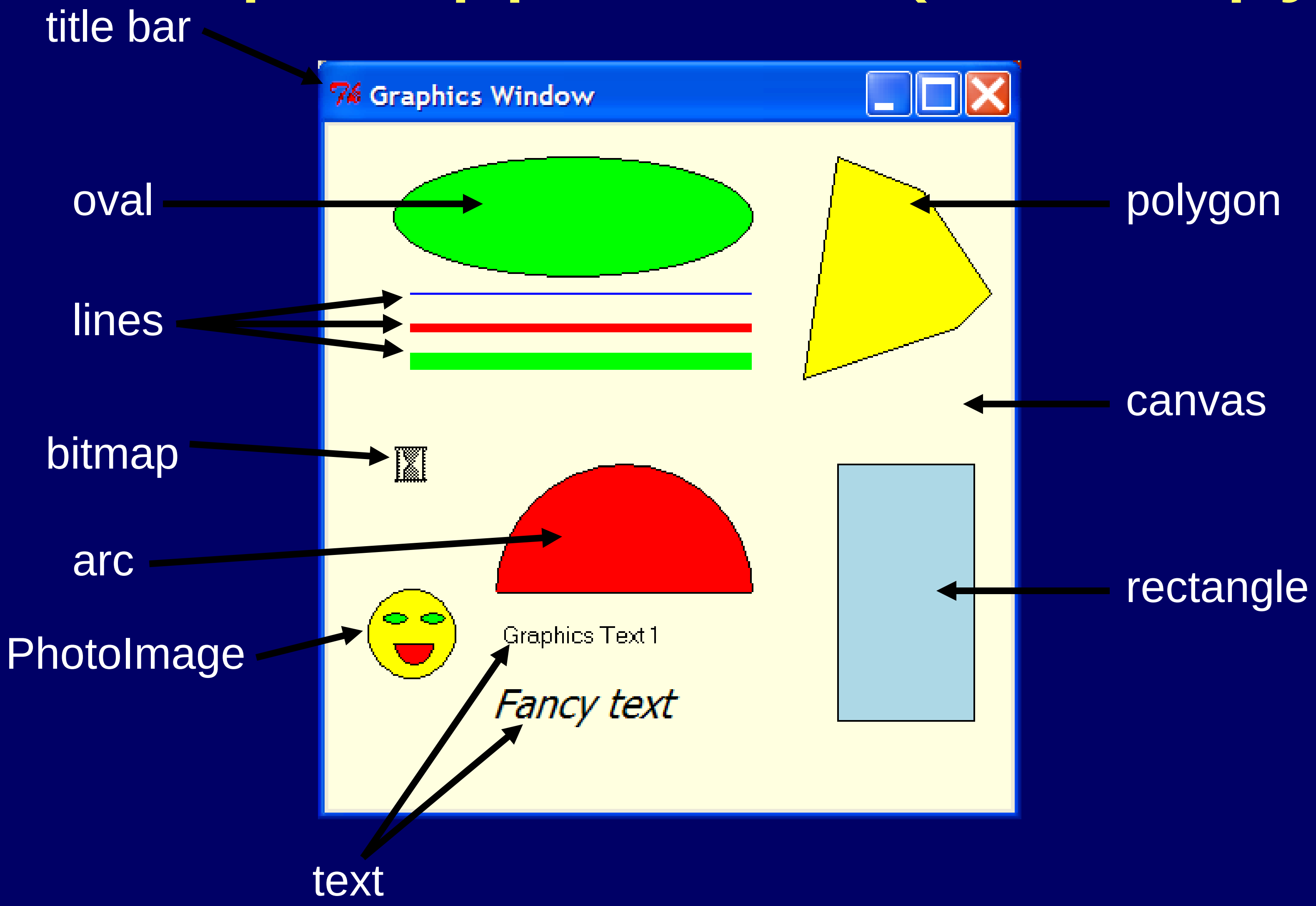
twobytwo.py (3)

```
label4 = Label(frame, \
               text = "label4", \
               width = 20, \
               bg = "yellow")
label4.grid(row = 1, column = 1)

root.mainloop()
```

Tkinter Drawing

- The Tkinter module provides widgets for drawing and implementing GUI applications.
- In this presentation we will concentrate on graphics.
- The next slide shows some of the graphics widgets we will discuss in this presentation. We will show the code that drew this shortly.

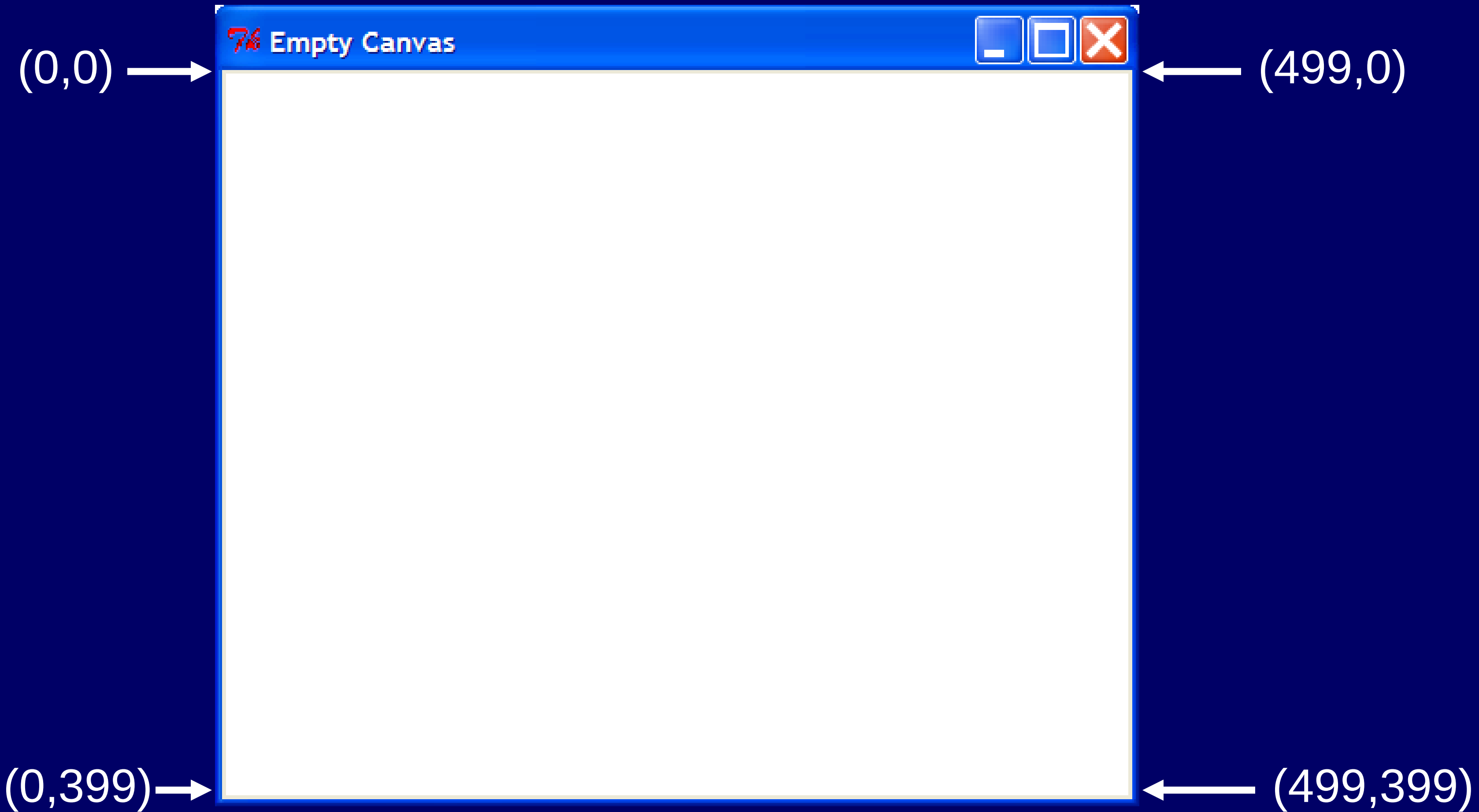
A sample application (draw1.py)



Graphics cookbook

- To create graphic applications, follow the following steps:
  1. Import the Tkinter module
  2. Create the root widget
  3. Create a canvas widget as a child of the root
  4. Use the pack manager to place the canvas in the root window
  5. Draw graphics images in the canvas
  6. Call root.mainloop()

Canvas Coordinate System
500 x 400 Canvas



Programming the Canvas

```
from Tkinter import *

# create the root widget
root = Tk()
root.title("Empty Canvas")
# create a 500 x 400 pixel canvas
can = Canvas(root, \
            width = 500, \
            height = 400,
            background = "white")
# place it at the TOP of the window
can.pack(side = TOP)
#enter main loop
root.mainloop()
```

Drawing lines

- In the discussion that follows, assume we have the following statement:

```
    can = Canvas(root, …)
```

- To draw a line, we use the create_line method:

```
    can.create_line(x0, y0, x1, y1 … )
```

- Options include:
  - **fill**—specifies the color of the line (default is “black”)
  - **width**—specifies the width in pixels (default is 1)

Line examples

- Draw a red line from (20,30) to (100,70):

```
can.create_line(20,30,100,70, \
                fill = “red”)
```

- Draw a green line with width = 5 pixels from (200,100) to (300,200):

```
can.create_line(200,100,300,200, \
                width = 5, \
                fill = “green”)
```

Drawing rectangles

- To draw a rectangle, use the create_rectangle method:

  `can.create_rectangle(x0,y0,x1,y1…)`

- **`x0,y0`** are the coordinates of upper left corner
- **`x1,y1`** are the coordinates of lower right corner
- Options include:
  - **fill**—specifies the color of the line (default is no fill)
  - **width**—specifies the width in pixels (default is 1) of the border.

Rectangle example

- Draw a rectangle with blue fill, upper left corner at (100,200) and lower right corner at (300,270):

```
can.create_rectangle(100,200,300,270, \
                     fill = “blue”)
```

Drawing arcs and pie slices

- To draw a pie slice, use the create_arc method:

```
   can.create_arc(x0,y0,x1,y1, \
                        start,extent …)
```

- x0,y0,x1,y1 are the coordinates of the rectangle that bounds the arc.
- The arc is drawn from the start angle (measured counterclockwise from 3 o'clock) to the start angle plus the extent
- Other options include:
  - **fill**–the fill color
  - **width**—the width of the outline (in pixels)
  - **style**—may be ARC, CHORD, or PIESLICE (default)

Pie slice example

- Draw a yellow-filled pie slice bounded by (100,200) and (250,300) from 0° to 180°:

```
can.create_arc(100, 200, 250, 350, \
               start = 0, extent = 180, \
               fill = "yellow")
```

Drawing polygons

- To draw a polygon with n vertices, use the create_polygon method:

  `can.create_polygon(`x_1`,`y_1`,…,`x_n`,`y_n`…)`

- The x_i, y_i are coordinates of the vertices. The coordinates should be given in order as they appear around the perimeter
- Options include
  – **fill**—the fill color
  – **outline**—the color of the border

Drawing ovals

- To draw an oval, use the create_oval method:

  ```
  can.create_oval(x0,y0,x1,y1…)
  ```

- x0, y0, x1, y1 define a rectangle that bounds the oval.
- Other options include:
  - **fill**—the fill color
  - **outline**—the outline color

Drawing text

- To draw text, use the create_text method:

```
can.create_text(x0,y0, \
                text = string …)
```

- x0, y0 are the coordinates of the text
- text is a string to be displayed
- Other options include:
  - font—the font used to display the text. Fonts are indicated as triples; for example,
    - (“Courier New”, 14, “bold”)
    - (“Tahoma”, 20, “italic”)

Inserting bitmap images

- The Canvas Bitmap item draws a 2-color bitmap on the screen:

```
can.create_bitmap(x, y, \
                  bitmap = “name” … )
```

- Options include
  - **foreground**—the color of the foreground pixels
  - **background**—the color of the background pixels
- Bitmaps available on all platforms include “info”, “error”, “question”, “hourglass”, and others

Inserting .gif images

- To insert a .gif image in the canvas, do the following:
  - Create a PhotoImage object
  - Use the create_image method to draw the image:

```
can.create_image(x,y, PhotoImage)
```

- Example:

```
photo = PhotoImage(file = “mypic.gif”)
can.create_image(100,200,photo)
```

Creating draw1.py (1)

```
from Tkinter import *

# create the root widget
root = Tk()
root.title("Graphics Window")

# create a 400 x 400 pixel canvas
can = Canvas(root, \
            width = 400, \
            height = 400,
            background = "light yellow")
# and place it at the TOP of the window
can.pack(side = TOP)
```

Creating draw1.py (2)

```
# draw 3 lines
can.create_line(50,100,250,100, \
                width=1, fill = "blue")
can.create_line(50,120,250,120, \
                width=5, fill = "red")
can.create_line(50,140,250,140, \
                width=10, fill = "green")

# draw a rectangle
can.create_rectangle(300, 200, 380, 350, \
                     fill = "light blue")
```

Creating draw1.py (3)

```
# draw an arc
can.create_arc(100, 200, 250, 350, \
               start = 0, extent = 180, \
               fill = "red")

# draw an oval
can.create_oval(40, 20, 250, 90, \
                fill = "green")
```

Creating draw1.py (4)

```
# draw a polygon
can.create_polygon(300, 20, \
                   350, 40, \
                   390, 100, \
                   370, 120, \
                   280, 150, \
                   fill = "yellow", \
                   outline = "black")
```

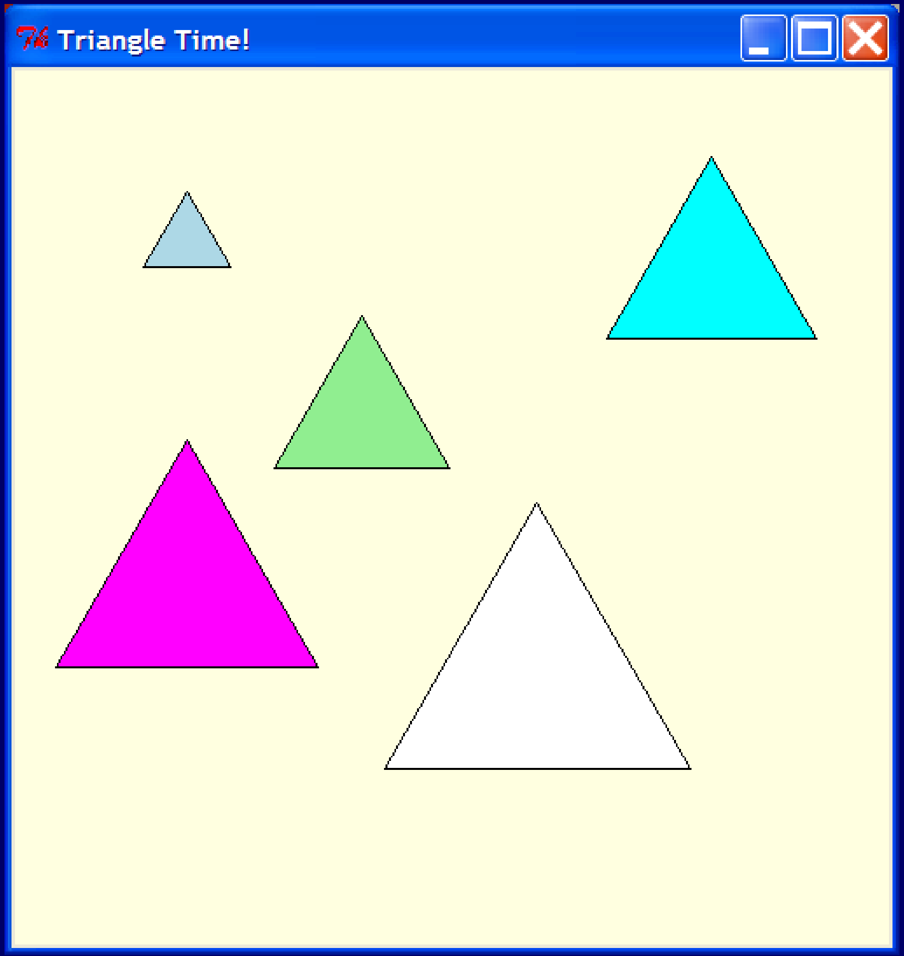
Creating draw1.py (5)

```
# insert a bitmap
can.create_bitmap(50, 200, \
                  bitmap = "hourglass")

# insert a PhotoImage
photo = PhotoImage(file = "happy.gif")
can.create_image(50, 300, image = photo)
```

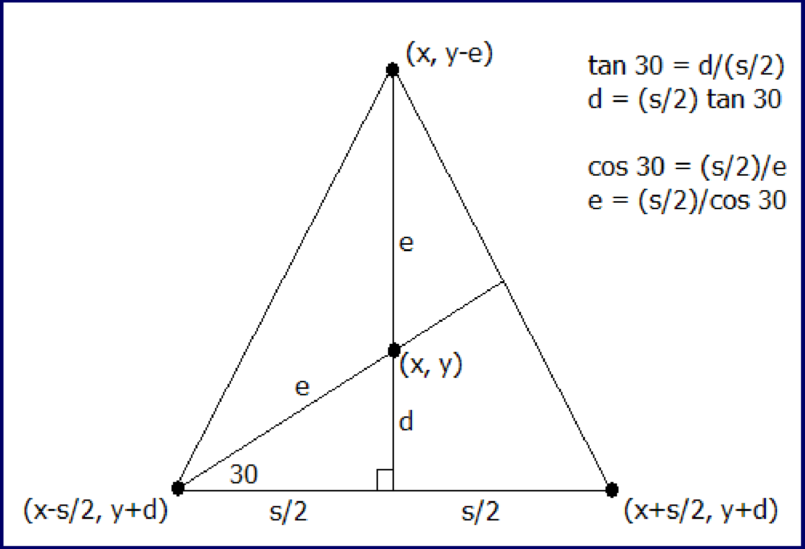
Creating draw1.py (6)

```
# draw some text
can.create_text(150, 300, \
                text = "Graphics Text 1")
can.create_text(150, 340, \
                text = "Fancy text", \
                font = ("Tahoma", 14, \
                        "italic"))

# enter main loop
root.mainloop()
```

Drawing triangles



Some geometry



triangle.py (1)

```
from Tkinter import *
from math import *

# triangle-drawing function
def triangle(c,x,y,side,fillcolor="white"):
    halfs = side / 2.0
    d = halfs * tan(pi / 6.0)
    e = halfs / cos(pi / 6.0)
    c.create_polygon(x, y-e, \
                     x+halfs, y+d, \
                     x-halfs,y+d, \
                     fill = fillcolor, \
                     outline = "black")
```

triangle.py (2)

```
root = Tk()
root.title("Triangle Time!")

can = Canvas(root, \
          width = 500, \
          height = 500, \
          background = "light yellow")

can.pack(side = TOP)
```

triangle.py (3)

```
triangle(can, 100, 100, 50, "light blue")
triangle(can, 200, 200, 100, "light green")
triangle(can, 100, 300, 150, "magenta")
triangle(can, 400, 120, 120, "cyan")
triangle(can, 300, 350, 175)

root.mainloop()
```