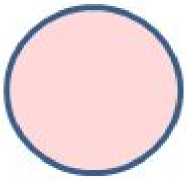
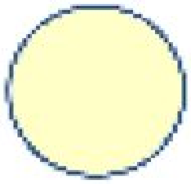
Activity 1.5.2

API for Tkinter Canvas

INTRODUCTION





Graphical user interfaces (GUIs) are made from **widgets**: buttons, sliders, and so on. The widgets are standard components of GUIs. Classes of widgets are created in code in a GUI toolkit. In this lesson you will create a GUI using the *Python*® programming language. We will use the Tkinter GUI toolkit, one of several GUI toolkits that are common across C++, *Python*, and other languages. One of the common widgets is a canvas. You use a canvas if you need to draw shapes.

Drawing on a GUI canvas is a little different than drawing shapes on images in the way you did in the last lesson. Canvas drawings are made out of objects. What’s the difference between drawing with objects and drawing with pixels? The two circles at the right had identical shapes when they were smaller. The top circle is stored in this document as a circle. If enlarged, it will still be a smooth circle. The circle on the bottom is stored in this document as an image–an array of pixel values. At the size you are probably viewing, it is already pixelated!

Is the image data for a TV show stored pixel-by-pixel or object-by-object? What about the data for a video game? How can you tell?

MATERIALS

- Computer with Enthought Canopy distribution of *Python*
- Graph paper with fine grid, for example 10 squares per inch

RESOURCES

1.5.2.PY StudentSourceFiles.zip

Reference Card for Tkinter.docx

Procedure

Greet your partner to practice professional skills. Set team norms for pair programming.

Launch Canopy. Open an editor window. Set the working directory to your folder. The IPython log is optional.

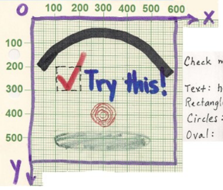
When running *Python*programs with Tkinter, there is an additional step that is necessary. Because only one main GUI can be running, Canopy needs to be taken out of its interactive mode.

a. In the Canopy Welcome window, select Edit > Preferences... from the menu at the top. In the Preferences dialog box that appears, select the ***Python*** tab. In the Python tab's window, from the dropdown selection for **Pylab backend**, select **Inline (SVG)**. If you have already launched a code editor window, You will be alerted that the *Python*kernel must be restarted for the change to take effect; if that alert appears, select **Restart kernel**.

b. Later, if you want to return to the interactive mode for programs not using Tkinter, repeat the above step but select **Interactive**

(Qt4) for the **Pylab backend**.

Consider the canvas design shown on the graph paper below on the left. On the right, the designer worked out the design (except the arc) using a coordinate system. We will use these numbers to implement the design using the methods of the `Canvas` widget.



Check mark : (100, 250) (150, 300) (220, 150)
width 20
Text: height 100pixels , center (380, 250)
Rectangle: (100, 200) (200, 300)
Circles: center (300, 400) radii = 10, 20, 30, 40, 50
Oval: bbox (100, 450) (500, 550)

a. Using your pen or pencil on the graph above, plot the three points whose coordinates are given for the check mark.

b. The designer's work indicates they think they have drawn the checkmark 20 pixels thick. Draw a line across the width of the check mark to indicate what is meant by 20 pixels.

c. Place a dot at the point (380, 250) indicated as the center of the text. Is the actual center of the text further to the left or further to the right?

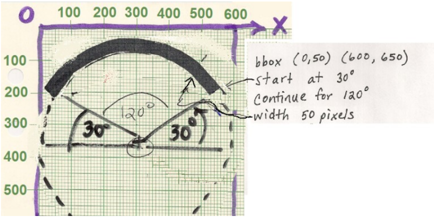
d. The list of radii shown above for the concentric circles' radii (10, 20, 30, 40, 50) can be produced with the `range (`*`start, stop, step`*`)` function. An example of `range()` is shown here.

```
In []: range(10, 16, 2)
Out[]: [10, 12, 14]
```

What arguments for `range()` will produce the list of radii? Check your answer in the IPython session.

e. The oval is specified by the coordinates of two points: the upper left and lower right of a **bounding box** . The **bounding box** circumscribes the circle and is specified with the (x, y) coordinates of opposite vertices. The designer noted (100, 450) (500, 550) for the oval's bounding box in the notes above. Sketch this bounding box in the figure above. Which of these coordinates from the designer's notes is off by a little bit?

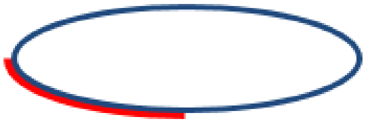
This designer made a separate diagram, shown below, to work out the coordinates and angles of the arc.



bbox (0,50) (600,650)
start at 30°
continue for 120°
width 50 pixels

a. Mark the two points specified for the bounding box in the figure. Are these points correctly identified?

b. The "start" of the arc is given as the angle in **standard position** : counterclockwise from the positive x direction. The "continue" is given as the **central angle** : the angle of the **sector** of the circle. Check your understanding of these terms by stating the starting angle and the central angle size of this arc shown in red. Then compare your answer with another team of two.

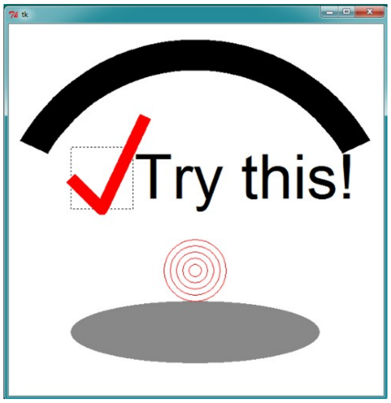


Open the file `TkinterDraw.py`.

GUIs are built with **widgets** like **buttons** and **sliders** . The program `TkinterDraw.py` is shown below. It creates a Canvas widget and then demonstrates some methods of the `Tkinter.Canvas` API that add objects to the canvas.

```
from Tkinter import * # Don't import like this except for Tkinter
root = Tk() # Create main window

# Make and place a Canvas widget for events and drawing
canvas = Canvas(root, height=600, width=600, relief=RAISED, bg='white')
canvas.grid()# Puts the canvas in the main Tk window

# Make four objects on the canvas
checkbox = canvas.create_rectangle(100, 200, 200, 300, dash=[1,4])
check = canvas.create_line(100, 250, 150, 300, 220, 150, fill='red', wi
message = canvas.create_text(380, 250, text='Try this!', font=('Arial',
shadow = canvas.create_oval(100,450,500,550, fill='#888888', outline='#

# Make an array of objects on the canvas
circles=[]
for r in range(10, 60, 10):
    circles.append(canvas.create_oval(300-r, 400-r, 300+r, 400+r, outli
# Make one more object on the canvas
canopy = canvas.create_arc(0, 50, 600, 650, start=30, extent=120,
                           width=50, style=ARC)

# Enter event loop. This displays the GUI and starts listening for even
# The program ends when you close the window.
root.mainloop()
```

a. Run the code. The output window should look like the one below and must be closed before the IPython prompt will come back.



b. For your note taking convenience and for your future reference, the following list shows some Canvas methods listed alphabetically:

Useful `Canvas` methods
Canvas(window, width, he [, options]) *Constructs a nev* *canvas object*
my_canvas.create_image(*x* *y1, image,* [, options]) *Creates an imag* *pasting it into the canvas*
my_canvas.create_arc(*x1,* *x2, y2* [, options])
my_canvas.create_line(x_1 x_2,y_2 [, …, x_n,y_n] [, options])
my_canvas.create_oval(x_1

x_2, y_2 [, options])
my_canvas.create_rectang y_1, x_2, y_2 [, options])
my_canvas.create_text(x_1 [, options])
grid() *Adds the canvas* *window's geometry manager*

c. The official documentation at **http://effbot.org/tkinterbook/canvas.htm**identifies options in keyword=value pairs. Using that documentation, identify what color the checkmark in line 13 would be if the `fill='red'` keyword-value pair had not been used.

d. The third party documentation at **http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/font** the 2-tuple used in the `create_text()` method: `font=('Arial', -100)`. What does the negative sign in -100 cause the method to do? What does the documentation say the method will do if the negative sign is omitted?

e. The methods listed in part b above all return the value of the `item_id` attribute for the object being placed on the canvas. You have to retain these for later use. Notice that each of these `item_id`s was stored in a variable. How many objects were created to put on the canvas?

List the variables storing those `item_id` numbers:

I. Tkinter doesn't give us direct access to the methods or attributes for those item objects enumerated in part e. But the `canvas` object knows about them. The `Canvas` class has methods that let us retrieve or set the attributes of the `canvas` items.

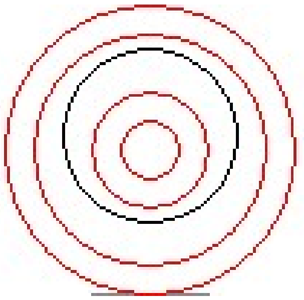
Methods to retrieve or set `Canvas` attributes

my_canvas.coords(*item*) *Gets the coordinates of the my_canvas item*
my_canvas.coords(*item*, x_1, y_1, x_2, y_2) *Sets the coordinates of the my_canvas item*

`my_canvas.itemcget(`*`item`*`,`*`keyword`*`)`
Get one option’s value for a canvas item

`my_canvas.itemconfig(`*`item`*`,`*`keyword=value`*`)`
Sets one option’s value for a canvas item

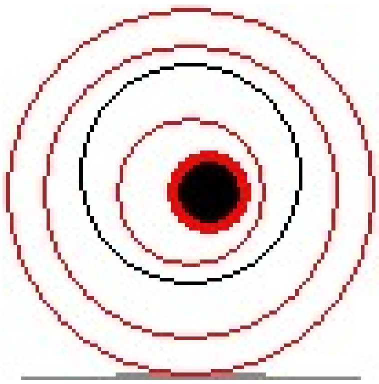
Insert the lines of code as shown below. The code must occur before `mainloop()`. The code will alter the concentric circles so that the middle circle is black and moved up 5 pixels, as shown below.



```
# Demonstrate changing a property of canvas' item.
canvas.itemconfig(circles[2], outline='black') # Change color
a, b, c, d = canvas.coords(circles[2]) # Get coordinates
canvas.coords(circles[2], a, b-5, c, d-5) # Change coordinates

# Enter event loop. This displays the GUI and starts listening f
```

Why do you think the code had to be inserted before `mainloop()` to have an effect?

f. Insert lines of code so that the inner circle is filled black, with a wider outline, and moved 5 pixels to the right:



Add an image to your canvas.

a. Insert the following lines of code before the `mainloop()` and run it. Your line numbers might be different.

```
# Get a filename in the same directory as this program
import os.path
directory = os.path.dirname(os.path.abspath(__file__))
filename = os.path.join(directory, 'canopyIcon.jpg')

# Open the image file and convert to an ImageTk object
import PIL.Image, PIL.ImageTk
img = PIL.Image.open(filename) # create a PIL.Image from the jpg fi
tkimg = PIL.ImageTk.PhotoImage(img) # convert the PIL.Image to a PI

# Add the ImageTk object to the canvas.
icon = canvas.create_image(tkimg, 150, 250)

# Enter event loop. This displays the GUI and starts listening for e
```

b. The code above will produce an error. Looking at the traceback, which line of code caused the error?

c. Use the official documentation for Tkinter's `Canvas` class (see Step 7c) to determine how to fix the arguments in your function call. *Hint: the arguments were in the wrong order, and one argument needs a keyword=value syntax.*

d. The official documentation for Tkinter includes more than the `Canvas` class. Refer to the official documentation at **http://effbot.org/tkinterbook/tkinter-index.htm**. At the bottom of **Part II: Class Reference**, follow the link for the `PhotoImage` class. According to the documentation, what types of image files can be created by the `PhotoImage` class without help from the `PIL` library?

e. The third-party documentation in Step 7d above is also part of a larger package of documentation for the entire `Tkinter` library. Refer to **http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/inde** see Section 8.6 on methods of the `Canvas` class. Read the explanation of the `tag_lower()` method. Predict what will result if you add lines 48-49 shown below. Then try it and finish the comment in line 48.

```
icon = canvas.create_image(150, 250, image=tkimg)

# Do what?
canvas.tag_lower(icon, check)

# Enter event loop. This displays the GUI and starts listening for e
```

CONCLUSION

One objective of this assignment was to increase your comfort level using online documentation. Describe something you like about using the "real" documentation, and describe something that is challenging about using real documentation.

How is a GUI canvas different than an image? Specifically, how do you think the data that stores information about the canvas is different than data that would store a similar-looking rectangular image with an identical appearance but as an image? Consider Steps 7f, 7g, and 8e in thinking about your answer.