Model-View-Controller Pattern with Tkinter

INTRODUCTION

Human-computer interaction (HCI) depends on the user knowing how to use the interface. A good interface is intuitive or easy to learn. Mice and touchscreens are now so common that users expect certain behavior tied to these devices in a graphical user interface (GUI).

Image courtesy Microsoft ©2011

How do programs usually respond to user input? You are familiar with what programmers have settled on as standard interfaces such as scroll bars ora dropdown menu. Such tools are often packaged in an API as widgets. How many more can you name?

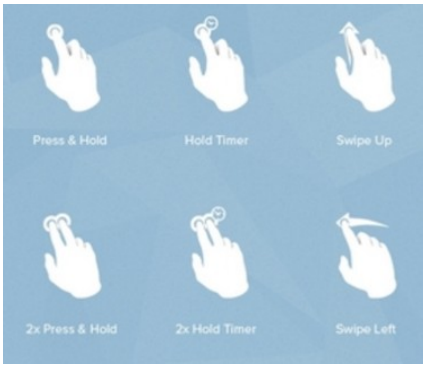


Image courtesy WebAppers ©2012

How do programmers make the interface behave the way a user expects it to? How do software developers design a solution to a problem so that it can be reused for other, similar problems?

MATERIALS

- Computer with Enthought Canopy distribution of *Python*® programming language
- Source files for Activity 1.5.3 and a teacher demonstration of the teacher source files for Activity 1.5.3

RESOURCES

1.5.3.PY StudentSourceFiles.zip

Reference Card for Tkinter.docx

Procedure

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

The central idea of this activity is that generalization allows reuse. Some problems appear over and over, often in completely different contexts. If you find the solution once, you can reuse the solution. Sometimes the solution needs to be generalized to solve all the problems it applies to.

What is a solution you developed to a real-life problem that you were able to reuse in a different situation?

Algorithmic problems often appear over and over. Algorithmic problems are problems that are solved by expressing an algorithm in human or computer language. For example, a classic problem is to sort a list. **Sorting a list** is an algorithmic problem, and there are several algorithmic solutions that can be precisely described. The different solutions can be compared to see which one is faster in various situations. Efficiency is usually an important criterion for algorithmic solutions.

Eight sorting algorithms are demonstrated at **http://www.sorting-algorithms.com/**. Observe each of these algorithms execute. Record the time used by each algorithm to sort the same 20-member random list. As shown below, you can click on the list displayed for any one algorithm, or you can race them against each other.

Which of these algorithmic solutions to the sorting-a-list problem is fastest?

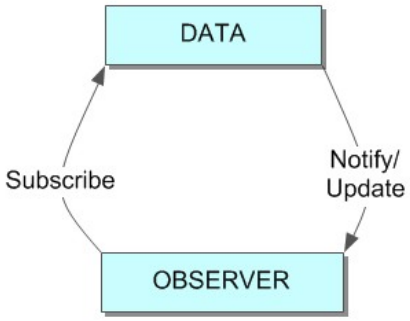
Describe the procedure followed by one of the sorting algorithms.

Another algorithmic problem that appears over and over in diverse situations is searching a list after it has been sorted. You are familiar with many instances of **searching a sorted list** . Fictional books in a library, for example, are already sorted by the author's last name. If you want to find a particular book, you follow a searching-a-sorted-list algorithm. The algorithm you follow can be reused to solve another instance of the searching-a-sorted-list problem.

What is another instance of the searching-a-sorted-list problem?

Software design problems also appear over and over. Many design problems are problems that are solved by creating a big-picture plan for a piece of software. These solutions are called **design patterns** . A design pattern guides software development, making it more likely that programmers will make rapid progress and avoid major roadblocks. You have seen that an object-oriented software solution is communicated by showing relationships among classes. Design patterns are even more abstract.

One design pattern is the **Observer** pattern, shown below.



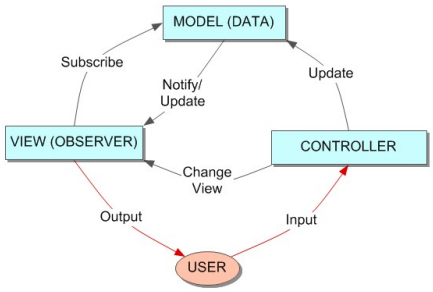
Data are stored. Observers can register to receive updates whenever the data change. The observer pattern applies to problems where several components need to know if data change. The pattern is also

used when several event handlers need to respond to one event. In a multi-player video game, for example, if one player moves, the movement should be shown on all the other people's screens that can see the player's new location.

If a component wants to know about events of a certain type, the component subscribes to that class of events. Different programming languages phrase this differently; you will also see this process described as a handler or listener, binding to or registering for the events. If an event in that category occurs, all the subscribers are notified by calling a method of the subscriber.

Consider a grade book system where parents, teachers, and students are emailed if a student misses an assignment. What is the event? What classes of objects might be subscribing to that event?

The **model-view-controller (MVC)** pattern builds on the observer pattern so that the user can use a controller to affect both the data and how the data are observed.



The model stores the data. The view presents data to the user. The controller lets the user change the view and/or the underlying data. Separating these three concerns was one of the key accomplishments of one of the first GUIs. This was the work of the Xerox PARC team and was captured and built on by both Apple and Microsoft.

Consider a grade book system and two students, Alice and Bob. Alice likes to look at her grades sorted by the due date, but Bob likes to look at his grades sorted by the percentage scores. The code for the grade book program is separated into model, view, and controller classes.

a. What are some data that the model would contain?
b. What classes of data might the programmers create?

c. What are some controls that the interface might offer the user?

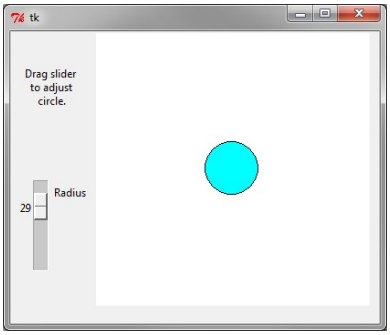
d. Describe how the user might view the data.

Part I: Event Handlers 1

We will consider the MVC pattern as we look at a GUI toolkit. There are several toolkits for building a GUI that can be used across multiple languages. For example, across *Python*, C++, and Java, there are the toolkits `Tk`, `wx`, and `Qt`. We will use `Tk`, which is implemented with the *Python*library `Tkinter`, so named because it is the *Python*interface to the Tk toolkit.

a. When running *Python*programs with Tkinter, Canopy needs to be taken out of its interactive mode to avoid having two GUI event loops competing with each other. 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 an alert appears advising you to restart the kernel, select **Restart kernel**.

b. Run `radius_changer.py`. The program is intended for a client who wants to visualize the distance represented by various pixel lengths by the video card and monitor. A **video card** is the hardware component of a computer which accepts data and instructions from the computer's processor and renders an image on the computer's monitor(s).



Describe what `radius_changer` does. Include the terms model, view, and controller in your description.

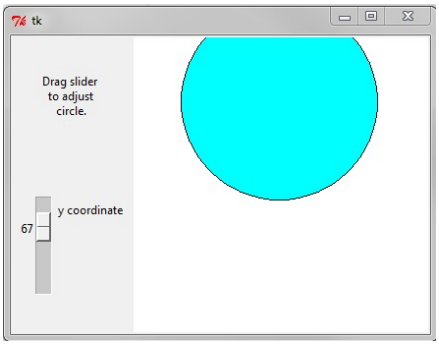
Analyze the `radius_changer` interface based on criteria for HCI discussed in an earlier activity.

The code for `radius_changer.py` is shown below. View Walkthrough #1 and refer to the *Reference Card for Tkinter*.

Circle key parts of the code and annotate with comments.

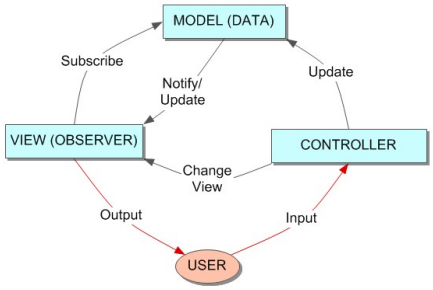
```
import Tkinter # often people import Tkinter as *

#####
# Create root window
####
root = Tkinter.Tk()

#####
# Create Model
######
radius_intvar = Tkinter.IntVar()
radius_intvar.set(100) #initialize radius
# center of circle
x = 150
y = 150

######
# Create Controller
#######
# Event handler for slider
 def radius_changed(new_intval):
    # Get data from model
    # Could do this: r = int(new_intval)
    r = radius_intvar.get()
    # Controller updating the view
    canvas.coords(circle_item, x-r, y-r, x+r, y+r)
# Instantiate and place slider
radius_slider = Tkinter.Scale(root, from_=1, to=150, variable=radius_intvar
                              label='Radius', command=radius_changed)
radius_slider.grid(row=1, column=0, sticky=Tkinter.W)
# Create and place directions for the user
text = Tkinter.Label(root, text='Drag slider \n to adjust \n circle.')
text.grid(row=0, column=0)

######
# Create View
#######
# Create and place a canvas
canvas = Tkinter.Canvas(root, width=300, height=300, background='#FFFFFF')
canvas.grid(row=0, rowspan=2, column=1)

# Create a circle on the canvas to match the initial model
r = radius_intvar.get()
circle_item = canvas.create_oval(x-r, y-r, x+r, y+r,
                                 outline='#000000', fill='#00FFFF')
#######
# Event Loop
#######
root.mainloop()
```

An adaptation of the previous program is shown below. The adaptation `position_changer` was demonstrated in Walkthrough #1. Describe the adaptation. Use the terms model, view, and controller.



Modify `radius_changer.py`to behave like `position_changer` by connecting the action of the slider to the circle's y-position instead of to the circle's radius. Save your code as directed by your teacher.

A token is a single element of syntax, like a reserved word or a variable name. Below, match each of the ten tokens listed on the right with a box or arrow in the MVC diagram on the left. Note that `Tk` is not limited to the MVC pattern. Also, one of the tokens doesn't match any element of the pattern!



- `IntVar()`
- `get()` method of `IntVar`
- `Scale()`
- `command` argument of `Scale`
- `variable` argument of `Scale`
- `Canvas()`
- item belonging to `canvas`like `circle_item`

- `coords()` method of `Canvas`
- `Label()`

Part II: User Stories

Another widget in `Tk` is the `Text` widget. We will use that widget for text output instead of using the Python's `print()` command. The `Text` widget is demonstrated with another adaptation of the previous program. View the demonstration of the adaptation `color_changer.py`. Describe the adaptation. Use the terms model, view, and controller.

`Tkinter` expects a color argument to be of type `str`. The string's first character is `#`. The next six characters specify hexadecimal digits: two digits each for red, blue, and green. In decimal, what is the color value represented by `'#A01145'` ?

red = green = blue=

Excerpts of code for `color_string_changer.py` are shown below. View Walkthrough #2.

Identify key parts of the code and annotate with comments.

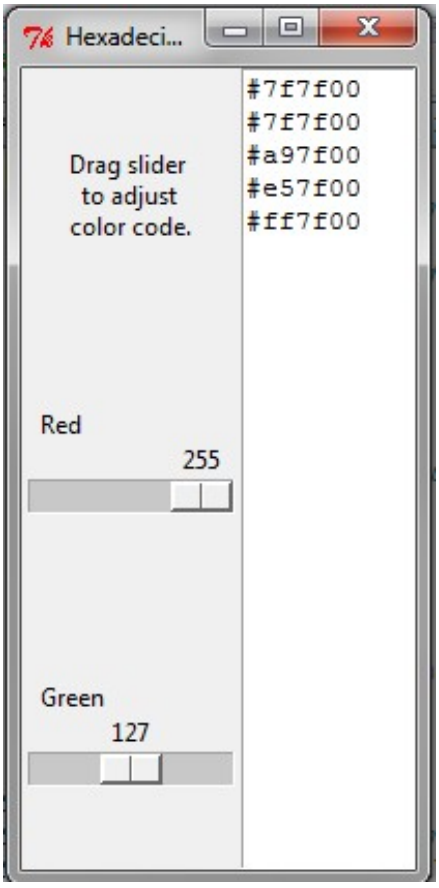
```
import Tkinter # Often people import Tkinter as *

#####
# Create root window
#####
root = Tkinter.Tk()
root.wm_title('Hexadecimal Explorer')

#####
# Create Model
#####
# Create two IntVar's and initialize them to 127
red_intvar = Tkinter.IntVar()
red_intvar.set(127)
green_intvar = Tkinter.IntVar()
green_intvar.set(127)

#####
# Create Controller
```

```
#####
# Event handler for slider
def color_changed(new_intval):
# Controller updates the view by pulling data from model
    editor.insert(Tkinter.END, '#' + \
                           hexstring(red_intvar) + \
                           hexstring(green_intvar) + '00\n')
    editor.see(Tkinter.END) # scroll the Text window to see the new bottom

# Instantiate and place sliders
red_slider = Tkinter.Scale(root, from_=0, to=255, variable=red_intvar,
                           orient=Tkinter.HORIZONTAL,
                           label='Red', command=color_changed)
red_slider.grid(row=1, column=0, sticky=Tkinter.E)
green_slider = Tkinter.Scale(root, from_=0, to=255, variable=green_intvar,
                             orient=Tkinter.HORIZONTAL,
                             label='Green', command=color_changed)
green_slider.grid(row=2, column=0, sticky=Tkinter.E)
# Create and place directions for the user
text = Tkinter.Label(root, text='Drag slider \n to adjust \n color code. ')
text.grid(row=0, column =0)

#####
# Create View
####
# Create a text editor window for displaying information
editor = Tkinter.Text(root, width=10)
editor.grid(column=1, row=0, rowspan=3)

#####
# Function to convert IntVar data to two hex digits as string
# for a Canvas widget color argument
#####

def hexstring(slider_intvar):
    '''A function to prepare data from controller's widget for view's consu
    slider_intvar is an IntVar between 0 and 255, inclusive
    hexstring() returns a 2-character string representing a value in hexade
    '''

    # Get an integer from an IntVar
    slider_int = slider_intvar.get()
    # Convert to hex
    slider_hex = hex(slider_int)
    # Drop the 0x at the beginning of the hex string
    slider_hex_digits = slider_hex[2:]
    # Ensure two digits of hexadecimal:
    if len(slider_hex_digits)==1:
        slider_hex_digits = '0' + slider_hex_digits
    return slider_hex_digits

#####
# Event Loop
#####
root.mainloop()
```

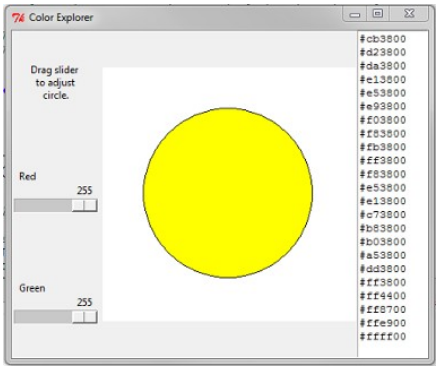
You will modify `color_string_changer.py` to behave like `color_changer`, which was demonstrated in Walkthrough #2. This modification can be thought of as a single backlog item. A backlog item is often stated as a user story, following a standard format: "___ wants to ___ so he/she can ___." Here is the user story completed by `color_changer`.

An artist wants to see how different RGB colors look so he/she can use them in a digital image.

Turn this into this Modifying the code to become `color_changer` will take several steps. Break the problem down into two or more tasks. Before modifying the code in the next step, describe the tasks here.



`color_string_changer`



`color_changer`

Task 1:

Task 2:

Task 3:

Modify `color_string_changer.py` to behave like `color_changer`, saving distinct versions along the way. Your files should include separate versions that accomplish individual

tasks you identified. Save your intermediate and final versions as directed by your teacher.

Part III: Event Handlers 2 (Variable Scope)

All GUIs are based on events and event handlers. `Tkinter` provides four ways to connect an event to an event handler. Three of them are:

- the `variable` argument of a widget
- the `command` argument of a widget
- the `bind()` of a widget

The `variable` argument doesn't let you write your own event handler; it just changes the value of the variable when the user uses the widget. The `command` argument lets you write your own event handler but only for the widget's built-in kind of event.
The `bind()` method, however, lets you write your own event handler *and* attach it to any event that exists. The events include mouse, keyboard, and timer events.

Two good sources of documentation on `Tkinter` events are listed here.

- **http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm**
- **http://infohost.nmt.edu/tcc/help/pubs/tkinter** Sections 52.2 - 52.6.

Lines 124-125 in the code excerpt below attach event handlers to two events on the `Canvas`. The events are `<Button-1>` and `<ButtonRelease-1>`. Refer to the documentation referenced above to identify the triggers for these events and describe them here. Use the find tool in your browser to find `<Button-1>`.

`<Button-1>`

`<ButtonRelease-1>`

Consider the following user story.

An artist is working on a series of acrylic paintings on large canvases. To plan each painting, she wants to paint circles on screen of different colors and experiment with their size, color, position, and overlap.

Run the program `canvas_circle_art.py`. Analyze the `canvas_circle_art` interface based on the sole criterion of accessibility.

Excerpts of code for `canvas_circle_art.py` are shown below. View Walkthough #4.

Identify key parts of the code and annotate with comments.

```
# Initialize globals so function defs can assign to them
startx, starty = 300, 300

# Define canvas' mouse-button event handler
def down(event): # A mouse event will be passed in with x and y attributes
    global startx, starty # Use global variables for assignment
    startx = event.x # Store the mouse down coordinates in the global varia
    starty = event.y

def up(event):
    tk_color_string = color(red_intvar, green_intvar, blue_intvar)
    r = (startx-event.x)**2 + (starty-event.y)**2 # Pythagorean theorem
    r = int(r**.5) # square root to get distance
    new_shape = canvas.create_oval(startx-r, starty-r, startx+r, starty+r,
                                   fill=tk_color_string, outline='#000000')
    shapes.append(new_shape)# aggregate the canvas' item

# Subscribe handlers to the Button-1 and ButtonRelease-1 events
canvas.bind('<Button-1>', down)
canvas.bind('<ButtonRelease-1>', up)
```

The `global` keyword in line 111 tells the *Python* interpreter to use variables from the global scope for assignment. The **scope** of a variable identifies which part of a program can use the variable name to refer to the variable's value.

a. What is the scope of `new_shape` as defined in line 119? Explain the implications.

b. Why isn't a `global` declaration needed in `up()` as it was in `down()`?

The `canvas_circle_art.py` program has been modified to create the following programs. Walkthrough #3 includes a demonstration of these programs.

Plan how to implement one of these solutions and complete your own version of the adaptation from `canvas_circle_art.py`. Save your program and documentation of your development process as directed by your teacher.

a. `canvas_rectangle_art.py`
b. `canvas_shape_art.py`
c. `canvas_recolor_art.py`

Part IV: Animation and Recursion

`Tkinter` usually accomplishes animation using recursion. **Recursion** is when a function calls itself. Recursion is one of the four building blocks of algorithms:

- Sequencing instructions
- Selecting instructions based on a conditional
- Iteration of instructions
- Recursion

As you discovered in Scratch, animation is created by moving a graphic element a small distance after each small time interval. In `Tkinter`, this is accomplished with the `after(msec, handler)` method of a `Tkinter` widget. The argument `msec` is an integer number of milliseconds, and the argument `handler` is the function name of the event handler.

The `after()` method creates a single timer. When the timer completes its time, it calls the handler only once. To call the handler over and over, we call the `after()` method again at the end of the handler, so the handler indirectly calls itself. When something references itself, surprising beauty and elegance can result.

Recursive algorithms can also be applied to images, as illustrated below.

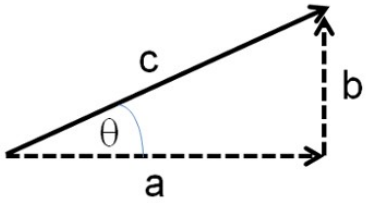
In general, recursion goes on and on until the algorithm encounters the **base case** . What is the base case in the recursion shown in this image? That is, when does it stop?

Run the program `bouncing_ball.py`.

a. View Walkthrough #4.

b. This code contains some trigonometry. Trig is common in code that animates or simulates a 2-D or 3-D environment with motion. Trig will only be used in this question. If directed to do so by your teacher, review Walkthrough #5

Now match the code's tokens on the right with quantities labeled in the figure on the left.



- `direction theta`
- `velocity_x a`
- `velocity_y b`
- `speed_intvar c`

Circle key parts of the code excerpt below and annotate with comments.

```
import math
def animate()
    velocity_
    velocity_
    canvas.mo
    x1, y1, x
    global di
    if x2>can
        direc
    if y2>can
        direc
    # Repeat
    canvas.af
animate()# ca
```

The `bouncing_ball.py` program has been modified to create `wrapping_ball.py`. This adaptation was demonstrated in Walkthrough #4. Plan how to modify the code to create your own version of this adaptation.

Create a program that performs like `wrapping_ball.py`. Save your program as directed by your teacher.

CONCLUSION

How is recursion different than iteration?

Algorithmic patterns and software design patterns both generalize a solution to a problem. Describe how these two types of generalized solutions are similar and how they are different.

Describe the process for designing a GUI