

Activity 1.5.3 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).  How do programs usually respond to user input? You are familiar with what programmers have settled on as standard interfaces such as scroll bars or a dropdown menu. Such tools are often packaged in an API as widgets. How many more can you name?  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? | clipped images,computer mice,computers,computing,cropped images,cropped pictures,icons,mice,mouses,PCs,peripherals,PNG,technologies,transparent background  Image courtesy Microsoft ©2011  Image courtesy WebAppers ©2012 |

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

Procedure

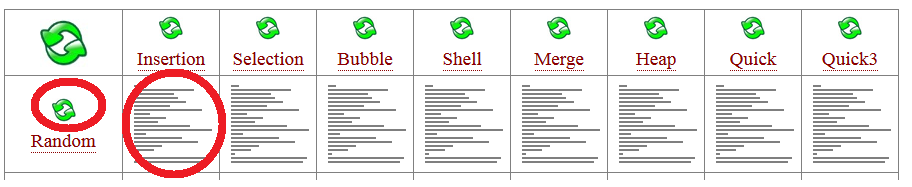
Im dumb and didn’t save the first section, you said it was fine

1. 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?

1. **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?

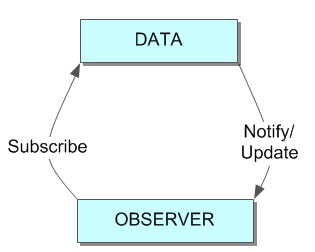
Describe the procedure followed by one of the sorting algorithms.

1. 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?

1. 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.

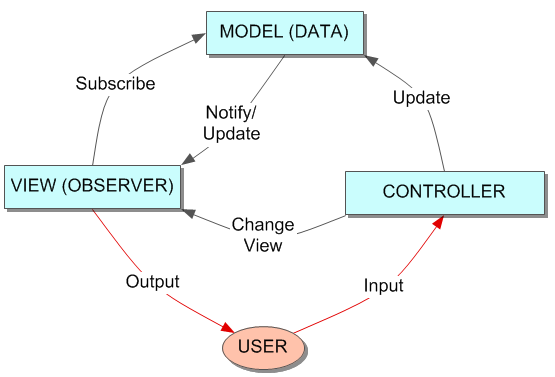


Data are stored. Observers can register to receive updates whenever the data change.

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?

1. 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.

1. What are some data that the model would contain?
2. What are some controls that the interface might offer the user?
3. Describe how the user might view the data.

**Part I: Event Handlers 1**

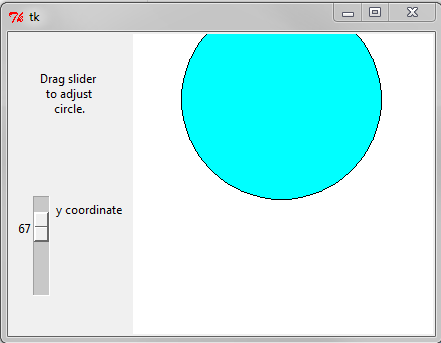
1. We will consider the MVC pattern as we look at a GUI toolkit. There are several toolkits for building a GUI that can be used across multiple languages.
2. When running *Python* programs with Tkinter, Canopy needs to be taken out of its interactive mode to avoid having two GUI event loops competing 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**.
3. 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. |

1. Analyze the radius\_changer interface based on criteria for HCI discussed in an earlier activity. Give a score and explain your reason (1 worst – 4 best)
   * + 1. Structure
       2. Simple
       3. Visible
       4. Feedback
       5. Tolerance
       6. Resuse
2. The code for radius\_changer.py is shown below. View Walkthrough #1 in the LMS and refer to the Lesson 1.4 Reference Card. Circle key parts of the code and annotate with comments.

|  |  |
| --- | --- |
| 9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | **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() |

1. 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.



1. 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.



Activity 1.5.3 Model-View-Controller Pattern with Tkinter

**Part II: User Stories**

1. 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.

The view lists the colors that have been selected

The model is temporarily holding and determining values for the colors from the sliders

The controller is in the form of two sliders, one for green value, the other for red

1. 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 =160 | green =17 | blue=69 |

1. Excerpts of code for color\_string\_changer.py are shown below. Identify key parts of the code and annotate with comments.

|  |  |
| --- | --- |
| 10  11  13  15  16  17  19  21  22  23  24  25  26  28  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  52  54  55  56  57  59  63  64  66  67  68  69  70  71  72  73  74  75  76  77  78  79  81  83 | **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 line*  *# 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*  **def** hexstring(slider\_intvar):  '''A function to prepare data from controller's widget for view's consumption  slider\_intvar is an IntVar between 0 and 255, inclusive  hexstring() returns a 2-character string representing a value in hexadecimal  '''  *# 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() |

1. You will modify color\_string\_changer.py to behave like color\_changer. 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 | color\_string\_changer | into this | color\_changer |

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.

Task 1: add the circle

Task 2: get the new color value for the cirlce

Task 3: have the circle update



Activity 1.5.3 Model-View-Controller Pattern with Tkinter

**Part III: Event Handlers 2 (Variable Scope)**

1. 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 (where is this found?)
* the command argument of a widget (where is this found?)
* the bind() method of a widget (how is this different than the top 2?)

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> left mouse click

<ButtonRelease-1> left mouse release

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 HCI criterion:

1. structure:

2. simple: 4- very simple to use

3. Visible: 3- decent positioning of stuff

4. Feedback: 4- visually see color change on most recent shape

5. Tolerance: 2- colors and sizes cant be changed after made, no way to clear screen

1. Excerpts of code for canvas\_circle\_art.py are shown below. Circle all variables, underline all methods**, put a box around event handlers and a line through events**.

|  |  |
| --- | --- |
| 106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125 | *# 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 variables*  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) |

1. 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.
   1. What is the scope of new\_shape as defined in line 119? Explain the implications.

In the up event

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

Uses the already global variables created in down, no other method needs th see its variables

1. The canvas\_circle\_art.py program has been modified to create the following programs which will be shown on the SMART board. Plan how you would implement one of these solutions and complete your own version of the adaptation from canvas\_circle\_art.py.

I want to see you take the canvas\_circle\_art.py code given to you, and on it next to each method state if it needs to change, and if it does, what within it needs to change.

You will need to use the resources given to you in order to complete this task.

Work in small chunks and test often.

Show me your finished program for one of the following:

1. canvas\_rectangle\_art.py
2. canvas\_shape\_art.py
3. canvas\_recolor\_art.py

^^^^^Already done?^^^^

**Part IV: Animation and Recursion**

1. 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

|  |  |
| --- | --- |
| The Droste effect – image recursion | 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?  Never(when frame length = 0) |

Conclusion

1. How is recursion different than iteration?

It changes and bases off itself( it modifies itself) rather than going through a sequence

1. Algorithmic patterns(the methods/steps to solving a problem expressed in a human/computer language) and software design patterns(guide software development often drawing the “big-picture” plan for a piece of software ) both generalize a solution to a problem. Describe how these two types of generalized solutions are similar and how they are different.

Software design patterns often use many algorithmic patterns to accomplist something. Algorithms are a smaller part of a software goal

1. Describe the process for designing a GUI.

Think about the layout you want

Try and make it

Settle for something that’s good enough☺