|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Introduction to Processing** |  |

|  |
| --- |
| **Your Tasks (Mark these off as you go)** |
| * Install processing * Have Ms. Pluska check your processing installation * Write your first processing program * Make circles * Explore the Sketch menu * Write the dealCards method in the CardDealer class * Call the dealCards method in the CardDealer class * Have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes * Complete challenges 1 thru 5 * Have Ms. Pluska check off your challenges 1 thru 3 before you continue * Receive credit for the group portion of this lab * Receive credit for the individual portion of this lab |

* **Install processing**

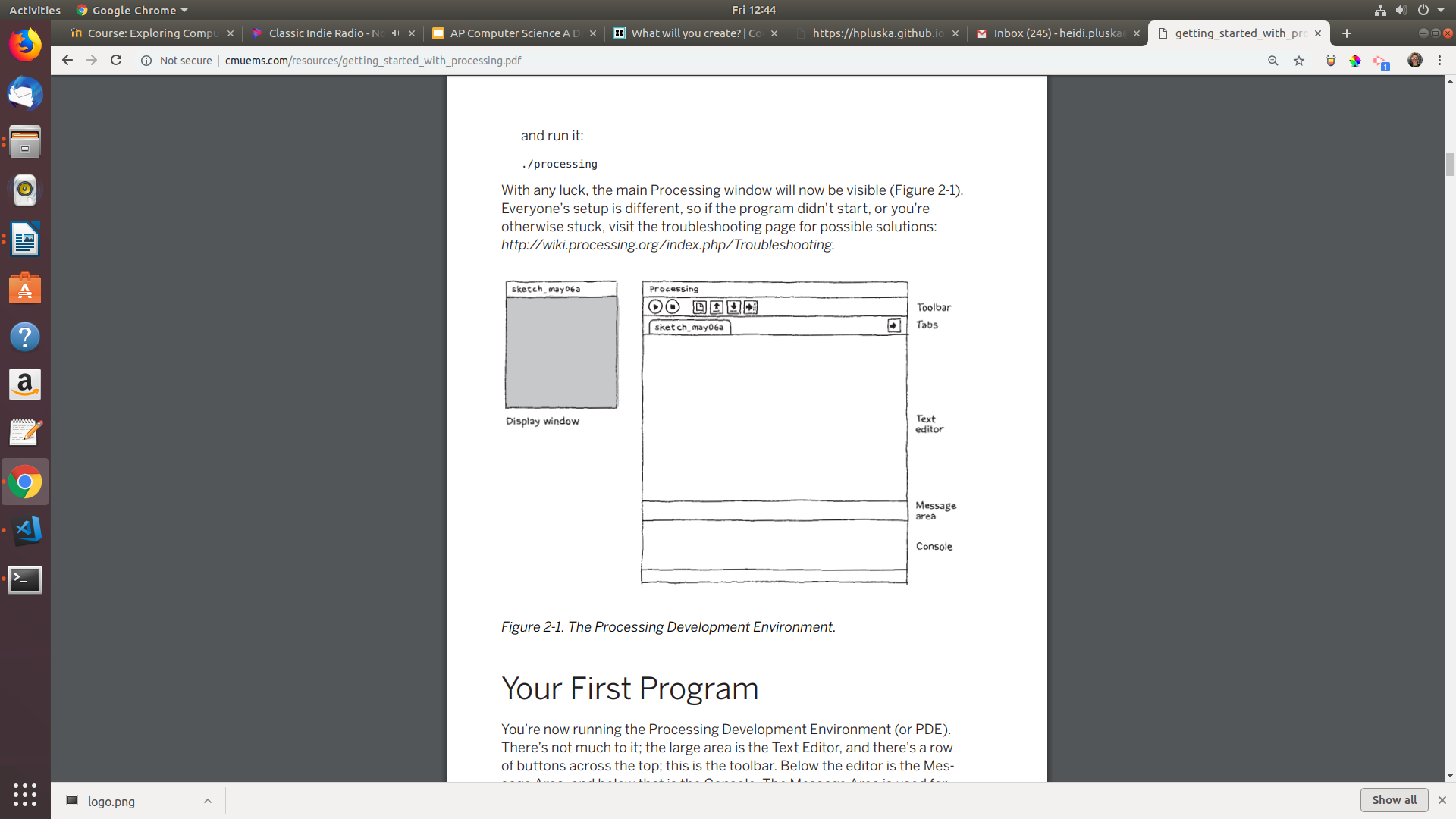
Start by visiting http://processing.org/download and selecting the Mac, Windows, or Linux version, depending on what machine you have. Installation on each machine is straightforward:

» On Windows, you’ll have a .zip file. Double-click it, and drag the folder inside to a location on your hard disk. It could be Program Files or simply the desktop, but the important thing is for the processing folder to be pulled out of that .zip file. Then double-click processing.exe to start.

» The Mac OS X version is a disk image (.dmg) file. Drag the Processing icon to the Applications folder. If you’re using someone else’s machine and can’t modify the Applications folder, just drag the application to the desktop. Then double-click the Processing icon to start.

» The Linux version is a .tar.gz file, which should be familiar to most Linux users. Download the file to your home directory, then open a terminal window, and type: tar xvfz processing-xxxx.tgz (Replace xxxx with the rest of the file’s name, which is the version number.) This will create a folder named processing-1.0 or something similar. Then change to that directory: cd processing-xxxx and run it: ./processing

With any luck, the main Processing window will now be visible as shown below. Everyone’s setup is different, so if the program didn’t start, or you’re otherwise stuck, visit the troubleshooting page for possible solutions: http://wiki.processing.org/index.php/Troubleshooting.



* **Have Ms. Pluska check your processing installation**



Before you continue have Ms. Pluska check your processing installation

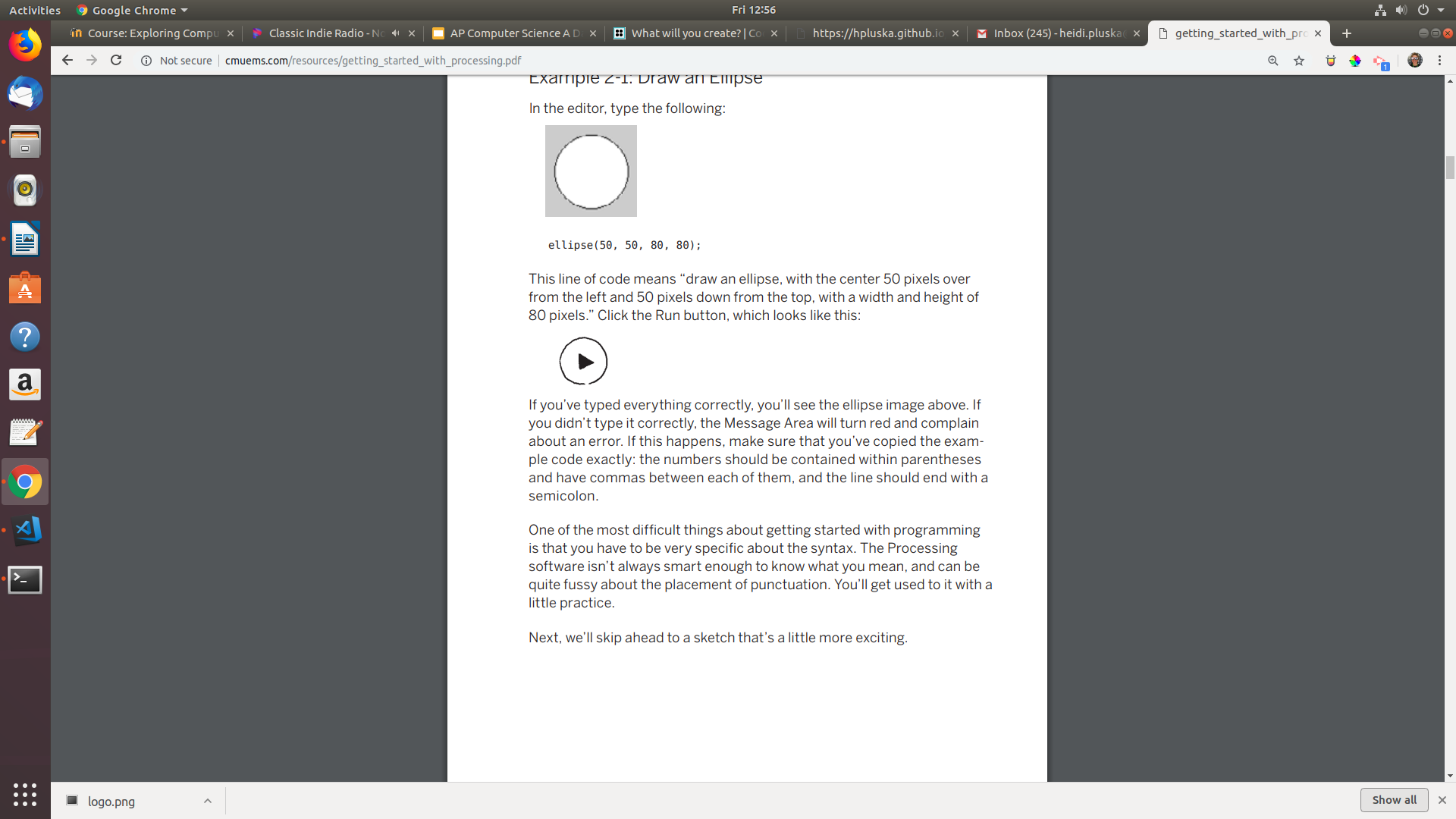
Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature \_\_\_\_\_\_\_\_\_\_\_\_

* **Write your first processing program**

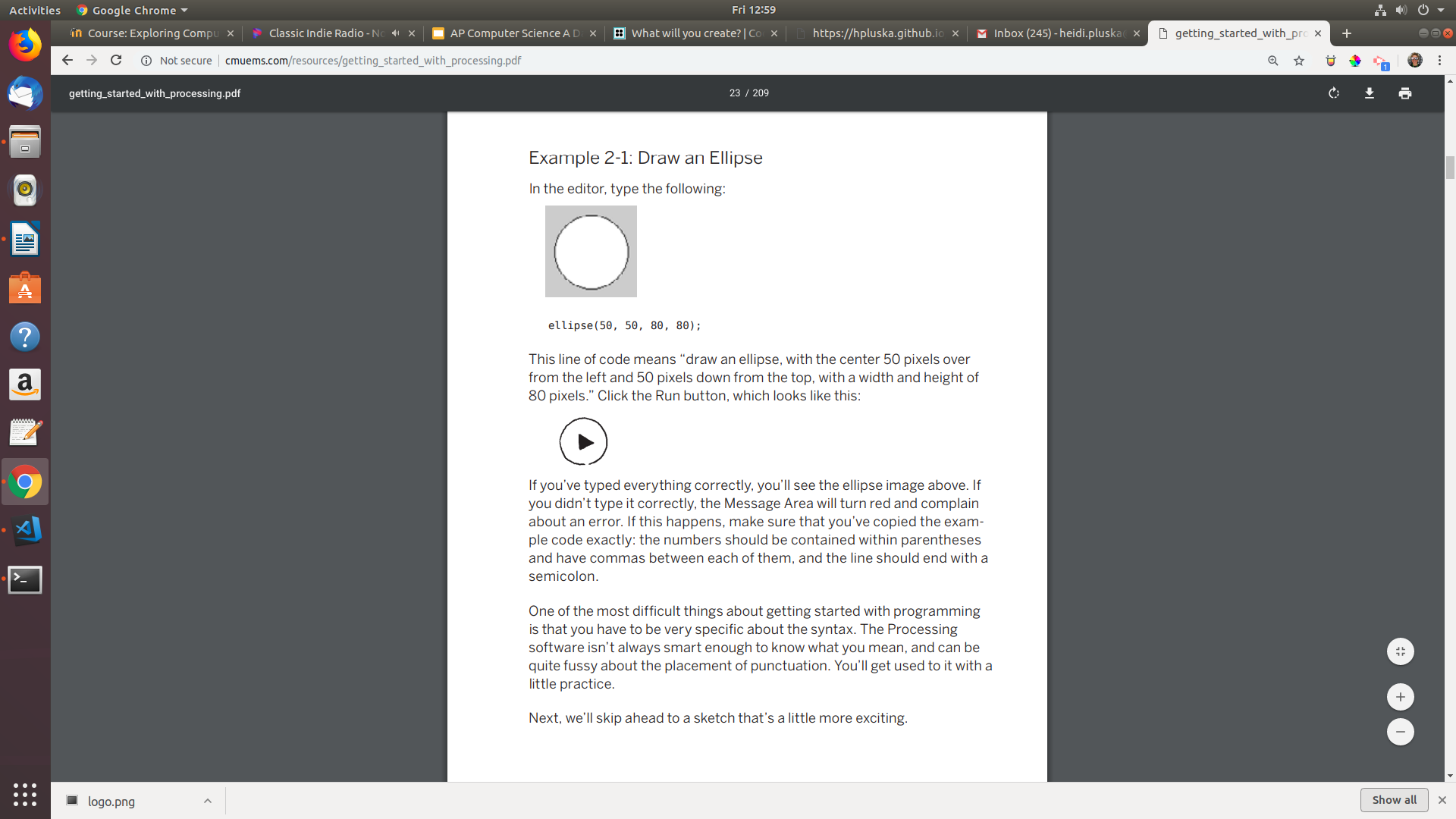
In the editor, type the following:

ellipse(50, 50, 80, 80);

This line of code means “draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels.” Click the Run button, which looks like this:



If you’ve typed everything correctly, you’ll see the ellipse image below. If you didn’t type it correctly, the Message Area will turn red and complain about an error. If this happens, make sure that you’ve copied the example code exactly: the numbers should be contained within parentheses and have commas between each of them, and the line should end with a semicolon.



* **Make circles**

Delete the text from the last example. Cut and paste the code below into the text editor.

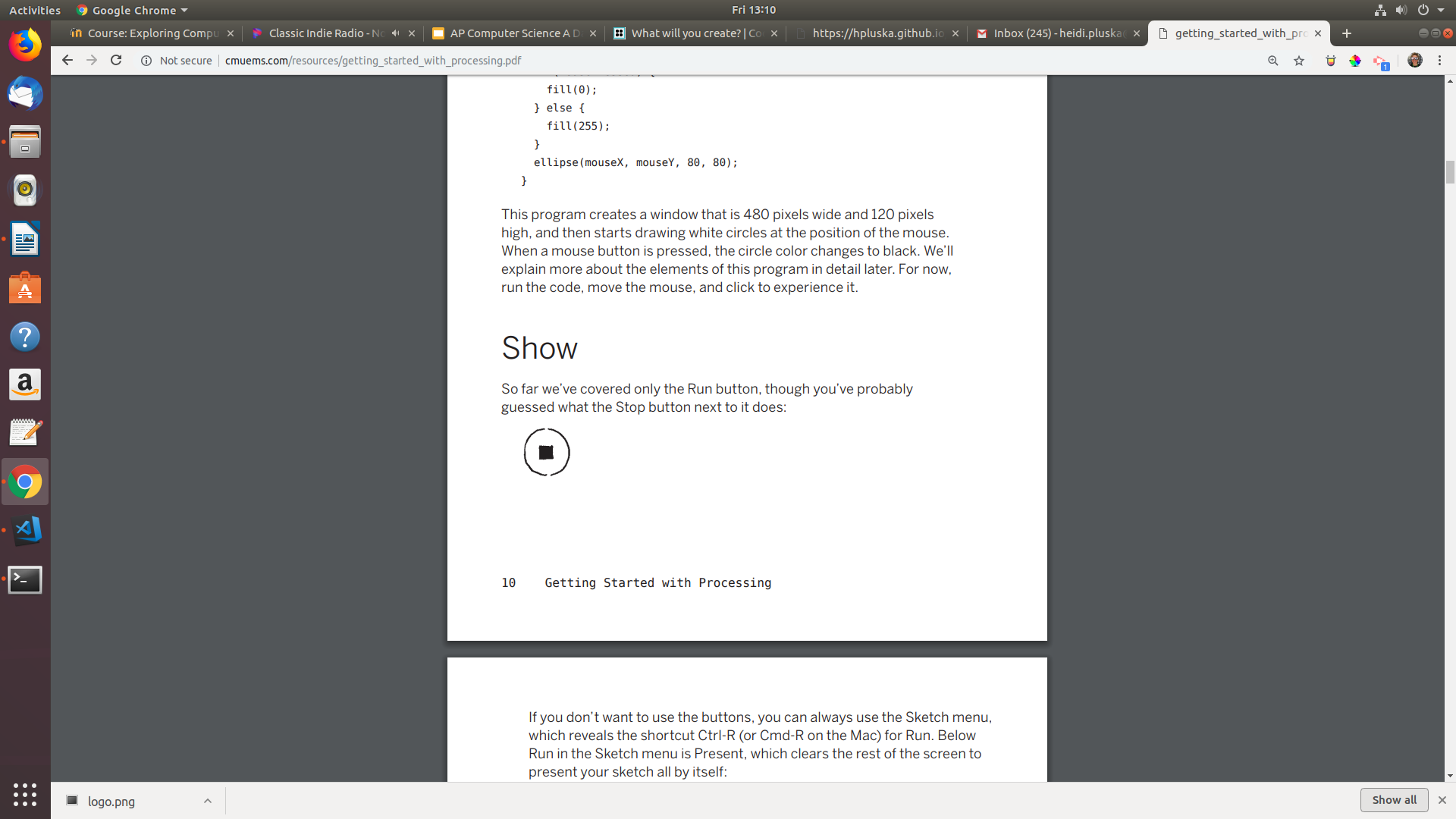
|  |
| --- |
| **Cicle code** |
| void setup() {  size(480, 120);  smooth();  }  void draw() {  if (mousePressed) {  fill(0);  } else {  fill(255);  }  ellipse(mouseX, mouseY, 80, 80);  } |

This program creates a window that is 480 pixels wide and 120 pixels high, and then starts drawing white circles at the position of the mouse. When a mouse button is pressed, the circle color changes to black. We’ll explain more about the elements of this program in detail later. For now, run the code, move the mouse, and click to experience it.

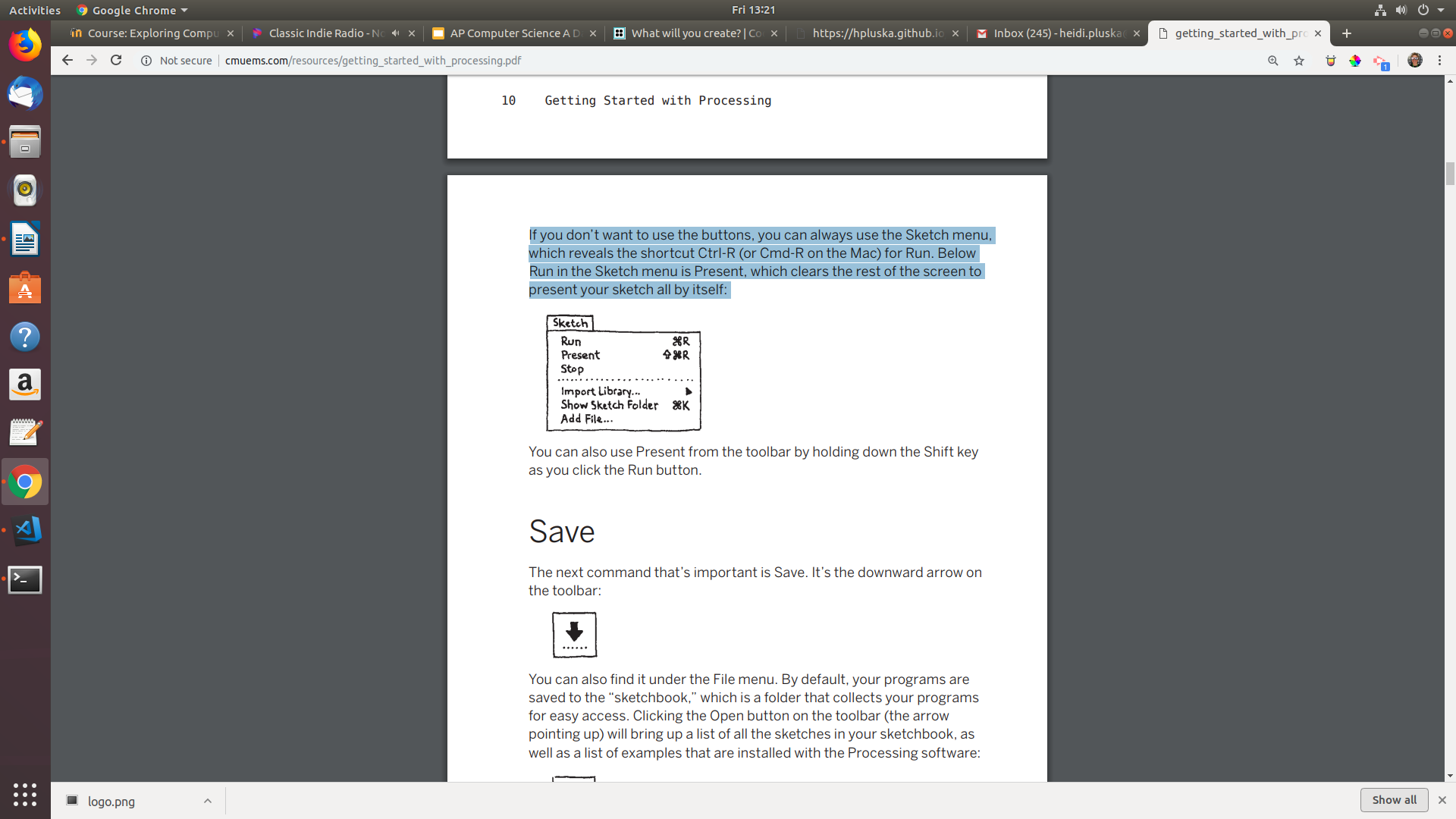
* **Explore the Sketch menu**

Show

So far we’ve covered only the Run button, though you’ve probably guessed what the Stop button next to it does

****

If you don’t want to use the buttons, you can always use the Sketch menu, which reveals the shortcut Ctrl-R (or Cmd-R on the Mac) for Run. Below Run in the Sketch menu is Present, which clears the rest of the screen to present your sketch all by itself:

****

You can also use Present from the toolbar by holding down the Shift key as you click the Run button.

Save

The next command that’s important is Save. You can also find it under the File menu. By default, your programs are saved to the “sketchbook,” which is a folder that collects your programs for easy access.

It’s always a good idea to save your sketches often. As you try different things, keep saving with different names, so that you can always go back to an earlier version. This is especially helpful if—no, when—something breaks. You can also see where the sketch is located on the disk with Show Sketch Folder under the Sketch menu

Open

The open command can be accessed from the file menu. It will bring up a list of all the sketches in your sketchbook, as well as a list of examples that are installed with the Processing software:

New

You can also create a new sketch by clicking the new command located under the file menu.

This will replace the sketch in the current window with an empty one. Holding down Shift when you press the New button will create a new sketch in its own window.

* **Write the main method in the CardDealer class**

Now that our Card and DeckOfCard classes are built, we can start dealing cards! Locate your CardDealer class and write a main method like shown below,

public static void main(String args[]){

}

Recall that we only want one deck of cards and that each deck contains 52 card objects. The static methods in the DeckOfCards class prevent us from confusing our current deck of cards, with a different deck of cards. To create a new deck of cards, we simply call the method in the main method of our CardDealer class,

DeckOfCards.buildDeck();

To see a particular card in our deck, we simply call the appropriate method. For example, the following code would show the value of the card at index 24.

System.out.println(DeckOfCards.showCard(24));

* **Declare the dealSize and dealt variables in the CardDealer class**

To play cards we need to know how many cards each play gets (dealSize), we also need to know the identies of the cards that have been dealt. To keep track of this information, declare the folllowing variables at the top of the CardDealer class,

private static final int DEALSIZE = 5;

private static Card[] dealt = new Card[DEALSIZE];

* **Write a the dealCards method in the CardDealer class**

Below the main method we will now write a new method. But, because this method will be used in the main method (which is static), it must also be designated as static. To get started, write the following,

|  |  |
| --- | --- |
| **dealCards method** | **Key terms defined** |
| public static Card[] dealCards(){    //leave some space here    return dealt;  } | Static – required because it will be accessed in a static method  Card[] - this method will return an array of dealt cards  dealt – the array of cards that will be returned |

To deal our cards will require that we populate the dealt array with the next card in the deck until it is full. This can be done with the code below. Add this code to the dealCards method you just wrote.

for(int i = 0; i < DEALSIZE; i++){

dealt[i] = DeckOfCards.nextCard();

}

Your final dealCards method should look as follows,

|  |
| --- |
| **Completed dealCards method** |
| public static Card[] dealCards(){    for(int i = 0; i < DEALSIZE; i++){  dealt[i] = DeckOfCards.nextCard();  }    return dealt;  } |

* **Call the dealCards method in the CardDealer class**

To deal your cards, simply return to your main method in the CardDealer class and write the following. This will deal a hand of cards.

dealCards();

* **Have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes before you continue**



Before you continue have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes

Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature \_\_\_\_\_\_\_\_\_\_\_\_

* **Complete Challenges 1 thru 3**

Challenge 1

Write a method called swapCards that swaps the values of two cards in the deck. The swapCards method should have the following signature,

public static void swapCards(Card a, Card b, int cardAIndex, int cardBIndex)

Challenge 2

The buildDeck method builds a sorted deck. The dealCards method deals the required cards. But, card dealers do not deal sorted cards. Your challenge is to write a method that shuffles the cards.

In the CardDealer class write the shuffleCards method, this method will have the following signature,

public static void shuffleCards()

Use the swapCards method you wrote in challenge 1 in this method. When you are done, call this method in the main method.

Challenge 3

Write a method in the CardDealer that finds the highest card in a shuffled hand and returns the card

* **Have Ms. Pluska check off challenges 1 thru 3**



Before you continue have Ms. Pluska check off challenges 1 thru 3.

Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature \_\_\_\_\_\_\_\_\_\_\_\_

* **Receive Credit for the group portion of this lab**

Make sure indicate the names of all group members, then submit this lab to the needs to be graded folder to receive credit for the group portion of this lab.

* **Receive Credit for the individual portion of this lab**

Implement challenges 1 thru 3 on your computer. Show Ms. Pluska the completed challenges to receive credit for the individual portion of this lab.