Digital Media Foundations
Assignment 4: Visualization in Processing

Deadline: Monday, 11/18, 9am (one hour before class begins)

Objectives:

1. Create a series of short computer programs, some of them interactive, within an *integrated development environment* (IDE).
2. Continue practicing the fundamentals of programming in a new language (including: statements, operators, variables, control structures, events, random numbers, parallelism, algorithms, and commenting)
3. Add some new programming ideas while deepening your knowledge of others (including: functions, data typing, file I/O).
4. Gain some understanding of how to work with external data files in Processing

Required software to Complete This Assignment:

- Processing (the tutorial linked below has instructions on downloading/installing the software)

Files to Turn In:
Processing Sketches (*.pde):

- ☐ Box and Line (A12): sketch1
- ☐ Mascot with Animated Text (B35): sketch3
- ☐ Racing Shapes (C10): sketch4
- ☐ Racing Shapes with Keyboard Control (C14): sketch5
- ☐ Draw With Mouse (C19): sketch6
- ☐ Random Shapes (C27): sketch7
- ☐ A New Function (C36): sketch8
- ☐ Shapes With a While Loop (C43): sketch9
- ☐ Manipulating Strings (D8): sketch10
- ☐ An External File (D32): sketch12

Processing Drawings (*.png):

- ☐ Mascot (B34): Mascot_file2
- ☐ Chart (D31): chart11

Compressing Your files to turn in: When you turn in your files for this assignment, we would like them to be compressed into a single zip file which you will upload to Canvas. The instructions for doing this are as follows:

- Windows Users:
  - Locate the folder that you want to zip using the Windows Explorer or search bar.
  - Right-click the file or folder, select “Send to,” and then select “Compressed (zipped) folder.” A new zipped folder with the same name is created in the same location.
- Mac Users:
  - Locate the folder that you want to zip in Finder.

- Right-click or command-click on the file to bring up the pop-up menu, then select Compress filename.
- The Mac will begin to compress the file or folder you've selected. Once it's done, you'll find a filename that ends in .zip right next to the file you selected.

NEW! color-coding in the assignment: we've included some color-coding in this document to help you navigate the assignment a bit better

Lab report: In this assignment, we have changed the color of all questions that need to be answered in your lab report to **RED**.

Code: yellow highlighting is used for code, to help you find and focus on the code quickly/differentiate it from its surroundings

Deliverables: blue highlighting designates a file you must save and turn in

Part A: Basics and the Processing Interface

OVERVIEW: In this section you will be learning the fundamentals of Processing. The Processing Web site offers up a number of tutorials on using the program. We will be starting off with some basics

1. **Open** up your web browser and navigate to:
   https://processing.org/tutorials/

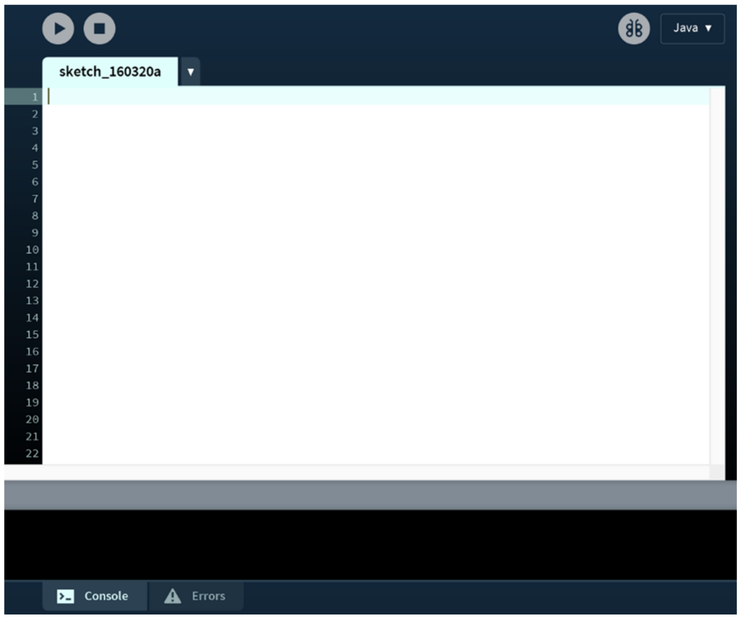
2. There are several tutorials on this page on learning Processing. To get some basics down **review** the "Getting Started" and "Processing Overview" tutorials. The "Getting Started" tutorial explains how to install Processing.

   OPTIONAL: If you have never programmed before this class and you found the last assignment challenging, you may want to ***instead*** complete the interactive video tutorial: "***Hello Processing***" at https://hello.processing.org/ (all six lessons) -- it is a tutorial designed for someone who has never programmed before. It takes about one hour.

TIP: Note that all **statements** in processing must end in a semi-colon (Processing statements are equivalent to "blocks" in Scratch). Processing **functions** – things with curly braces { } -- do not need to end in a semi-colon (Processing functions are mostly equivalent to "scripts" in Scratch).

3. In the Processing Overview, did you notice the authors reference writing "a 'Hello World' program?" Now that you're a programmer you will remember what "a 'hello world' program" means from Scratch! Recall it's a synonym for "the simplest program" or "the first program you write" in a new programming language. What is a "hello world" program in Processing? **This will be a response in your lab report.**

4. You now have some rudimentary skills that can be useful in writing Processing programs. Let's review some of this knowledge. If it isn't started already, click the Processing icon (Mac) or double-click "processing.exe" (Windows) to **start the application**. You'll see the Processing Development Environment (PDE):

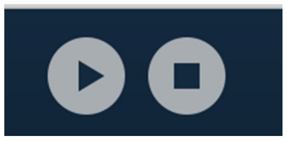


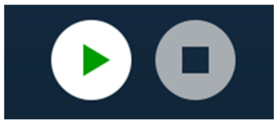
5. This is the editor you will be using to write your programs. **Notice the two tabs** at the bottom: "Console" and "Errors". Console will display any information that you print from your program, while "Errors" displays any syntax/logic problems with your code. The message area is the grey bar just above the "Console" and "Errors" area.

6. In the text editor, **type "Your code goes here"** then wait a moment. What do you notice in the message area of the Processing Window? **This will be a response in your lab report.**

7. **Click on the error tab**. What do you now see in the Console? **Enter your response in the lab report**.

8. In the toolbar you should see a "Play" and "Stop" button. **Hover over the buttons** to see their labels ("Run" "Stop"). Play is used to run your program, and stop (as you might expect) stops the program. These are equivalent to the Green Flag and Red Stop icons in Scratch. When your program is running, the play icon will be green.





9. The output of your code goes in the Canvas (the Stage in Scratch). Unlike Scratch, in most cases, you get to determine the attributes of the Canvas as part of your code. You define the Canvas using the setup() function. Remove all of the text from the editor window and **type the following code** [we recommend you do not copy and paste]:

```
void setup() {
        size(400, 400);
}
```

Then click Play (a.k.a. "Run"). You should now have a Canvas window with 400 x 400 pixels. The size of the canvas is determined by the numbers you enter into the size command

DOUBLE-CHECK: Note the use of curly brackets (blocks). Block functions (meaning functions with curly brackets after them) often require the following Processing syntax.

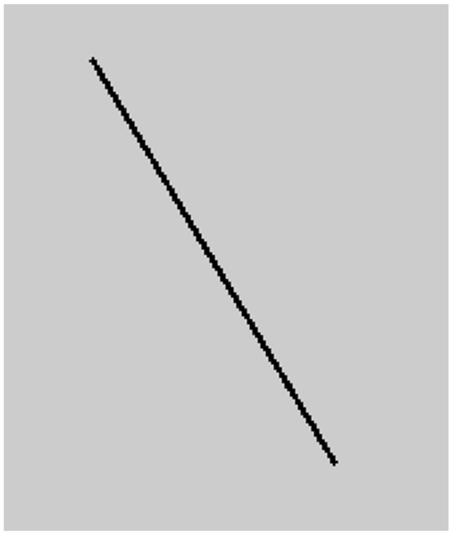
```
    void functionName(parameters) {
            Your code goes here
    }
```

Some functions do not include parameters within the parenthesis after the function name. Examples are setup(); and draw(); Some do not require anything before the functionName.

10. Recall in Scratch that the "Forever" block was used to enter code that you wanted to run repeatedly. In Processing the draw() function occupies a similar role. It is used to draw things on the Canvas. Let's start with drawing a line. In the text editor, **type the following code** below the setup() function you wrote in the previous step:

```
void draw() {
        line(100, 100, 190, 250);
}
```

Now click **Play** ("Run"). You should have a line that looks like the one to the right.



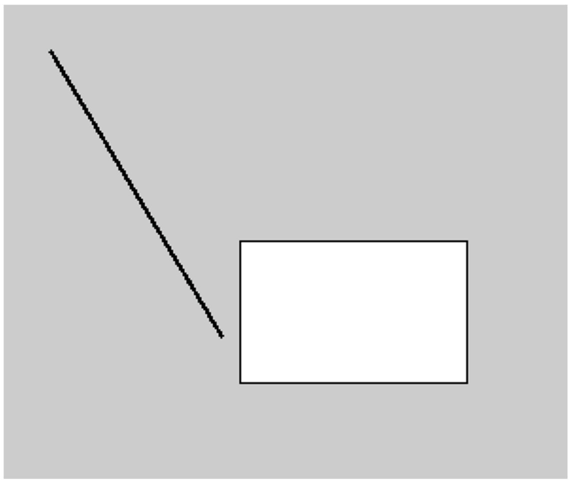
Note the angle of the line is sloped. This is because drawing a line in Processing takes the format line(x1, y1, x2, y2), where (x1, y1) and (x2, y2) are the x and y coordinates of both endpoints of the line. In Processing the origin coordinates (0, 0) of the canvas is in the topmost left corner, while the largest coordinates, in this case (400, 400), would be the bottom-most right corner of the Canvas. This is important to know as it will help you determine what coordinates to use when you draw shapes.

PITFALL: Note that Canvas uses a different x,y coordinate system than Scratch does. In Scratch 0,0 is the middle of the stage, but in Processing 0,0 is the upper-left corner.

11. Now let's draw a rectangle. In the text editor within the same draw() function **add the following new line** *inside* the draw() curly brackets { }:

rect(200, 200, 120, 75);
Now click play (run).

Your Canvas should now look like this image to the right (not to scale).



Rectangles have something in common with lines. The first two numbers represent the (x,y) co-ordinates of the rectangle's starting point. However, the third number represents the width of the rectangle (its length on the x-axis), and the fourth number represents the height (its length on the y-axis).

OPTIONAL: We will be drawing more shapes in Part B, but if you still need some clarity with shapes, review the Processing tutorial, "Coordinate System and Shapes"

12. **Save** the program you have just created using the following filename, ***myUMuniqname_sketch1***. To save the file go to the menu at the top of the screen and click "File" > "Save As." This will be the 1st file that you submit for this assignment.

DID YOU NOTICE: The basic structure of shapes in Processing is the same as it is in *SVG graphics* which we covered way back in Assignment #1? Many terms are identical: SVG has <ellipse> while Processing has ellipse(). SVG has <rect> while Processing has rect().

PRO TIP: You can draw shapes in Inkscape then import them into your processing Sketch. The Processing function to load a graphic from an external .svg file is loadShape(). If you want to try this, be sure to save from Inkscape as "plain" SVG.

PART B: Drawing a Mascot in Processing

Overview: In this section, you will be drawing a "mascot" following step-by-step instructions. The point of this exercise is to get you more comfortable with the basics of Processing. Be prepared to do some independent work in the last part of this section.

1. **Start a new sketch** in the Processing Development Environment (PDE). A fast way to do this is "File" > "New". The editor should be blank.

2. In the editor **create a canvas** with the following size (500, 650). Review Part A, Step 8 if you need some assistance writing the code for this step.

TIP: In the early stages of learning to program, it might be a useful technique to run your program after each new line of code. This can help you catch errors immediately. This is sometimes called ***iterative design***.

3. Next we will write the draw() function. In this step we will also change the background color for our Canvas. In the text editor, create the draw function and type the following code inside. **See A9 if you need a refresher on the draw() function.**

```
background(#FFCB05);
```

4. We also want to start including some comments with our code. Processing comments begin with //. Here is a comment for the above code

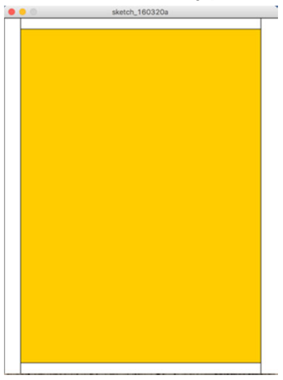
```
background(#FFCB05); // This is the color code for Michigan's Maize color
```

Type an appropriate comment for the background command. Do not simply copy the comment above.

5. Next, let's **draw** some rectangles that will be used as a frame for our Mascot. Type the following code into your draw function:

```
rect(0,0,30,650);
rect(470,0,30,650);
rect(30, 630, 440, 20);
rect(30, 0, 440, 20);
```

Click Run. If you have followed all the steps correctly, your Canvas should now look like the image to the right (not to scale):



6. If the previous step with the background color didn't clue you in, the background for this drawing will be using Michigan's colors. We have the Maize but the blue is missing. Let's **add some color** to the rectangles we just drew. **Type** the following line of code inside the draw function.

```
fill(#00274C);
```

7. When you added that fill() statement, where did you place it inside the draw() { } function? At the top or the bottom? Note that it still works if you add it at the bottom because draw() – just like the "Forever" block in Scratch – runs continuously from top to bottom until the program is stopped. But if we add additional fill() statements your code may break because the statements will override each other.

If you put the fill() command *after* drawing the rectangles with rect(), this means that they were drawn in the default color (white) and then the color was changed using fill(). The computer does this too quickly to see the white change to blue. If you put the fill() command *before* drawing the rectangles, the computer draws blue rectangles.

Since the latter is more elegant and makes more sense (there is no need to draw the white rectangles if we never want them), **move your fill() statement so that it is above the rect() statements** that draw the rectangles. (If it is already above them, you can leave it alone.)

COMMON PITFALL: It is important to write code that makes sense to future programmers (including your own future self) even if the result doesn't change the user's experience. That is why putting fill() above rect() is correct in this case even if the program's behavior for the user does not change in any detectable way.

8. Add at least **two appropriate comments** to your code.

9. We now want to begin drawing the mascot. Let's draw some ears for this mascot. Now in your draw function **type** the following:

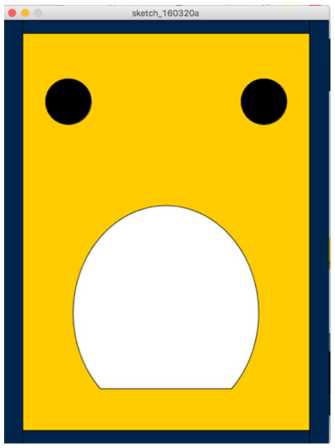
```
fill(0);
ellipse(400, 125, 70, 70);
ellipse(100, 125, 70, 70);
```

   We have just introduced a new shape, ellipse which is used to draw circles. If you reviewed the Coordinate System and Shapes tutorial you should already be familiar with the format. Note simply that it is similar to a rectangle taking the format (x, y, width, height). The major difference is the (x,y) starting/reference coordinate for a rectangle is the uppermost corner, while the starting/reference coordinate for an ellipse is the center.

10. Now we want to draw the body for the mascot. **Type** the following code:

```
fill(255);
arc(250, 450, 285, 330, PI-QUARTER_PI, TWO_PI+QUARTER_PI, CHORD);
```

   Your canvas should now look like the image to the right:



   This code introduces CONSTANTS. In Processing, constants are written in capital letters. Conceptually they are related to variables. While variables may change when the program runs (for example, the processing variable **mouseX** represents the position of the mouse right now), but constants are always the same. Processing constants are **PI**, **TWO_PI**, **HALF_PI**, **QUARTER_PI**, and **TAU**. You can think of these as a way to save on typing 3.14159265358979323846 (a.k.a. **PI**) so often!

   This code also introduces OPTIONS (in Processing also called "modes" or "parameters"). In the code above the word **CHORD** is an *option* as it is a choice among a fixed number of choices (**CHORD**, **PIE**, or **OPEN**) that specify the different ways that the function arc() might work.

PITFALL: There are also other things in processing that are capitalized, so not all capitalized things are constants or options.

*********** EXTEND YOUR KNOWLEDGE**********

We have also introduced another shape – the arc. Arcs are a special type of ellipse. The format is as follows: arc(x, y, width, height, start, stop). The main difference is the addition of two new criteria – the start and stop positions, which are measured in radians. The Start position is always the 3 o'clock point of your circle and goes clockwise around. You may recall from geometry that a circle is equal to 2 * PI. Which means the 6 o'clock point would be ½ PI, the 9 o'clock point PI, the 12 o'clock point 1.5 PI and so forth. Processing has built in variables to capture various points across the arc (QUARTER_PI, HALF_PI, PI and TWO_PI). In this case we need the arc to start midway between the 6 o'clock and the 9 o'clock points which can be represented as PI – QUARTER_PI (as in subtract a ¼ PI from PI). Then we also needed it to go around the circle and end at the ¼ PI position. The Chord creates the line at the bottom. This is an optional variable. Other options include (Open, and PIE).

PRO TIP: You can also use multiplication and division signs to represent angles. E.g. the starting point for the above arc could be written as "PI*.75" and the stop position as "PI*2.25"

QUESTION: What happens if you simply enter QUARTER_PI in the stop position? Enter your answer in the lab report now.
Based on the explanation above, why do you think this doesn't work? Explain in the lab report.

11. Let's move on to draw the head of the mascot. Type the following code:
```
ellipse(250, 200, 300, 300);
```

12. **Notice the order** of the drawing code. We did not start with the head even though that might be a common starting point for drawing. The reason is simple – the last code written superimposes over previous code. In our drawing we need the circle of the head to be drawn over the body and the ears of the mascot which is why it is drawn last.

13. Time to start filling out some additional details such as eyes and paws. This step should go quickly. **Type** in the following code.
```
fill(0);
ellipse(330, 180, 75, 75);
ellipse(170, 180, 75, 75);
arc(335, 420, 80, 100, PI-QUARTER_PI, TWO_PI+QUARTER_PI, CHORD);
arc(165, 420, 80, 100, PI-QUARTER_PI, TWO_PI+QUARTER_PI, CHORD);
```
This should have created eyes and paws for the Mascot. Is it taking shape yet?

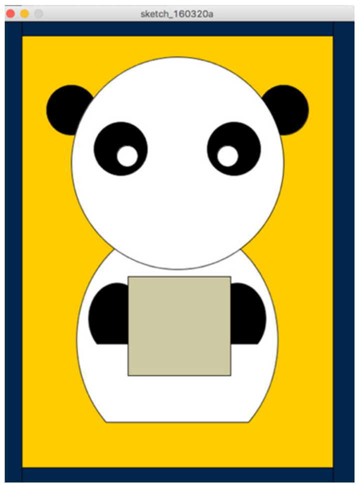
13. We want to add some details to the eyes. Now **type** in the following code

```
fill(255);
ellipse(321, 190, 30, 30);
ellipse(179, 190, 30, 30);
```

14. Our Mascot needs a poster. **Type** the following

```
fill(#CDC9A5);
rect(180, 360, 145, 140);
```

If you have followed all the steps correctly, when you run (Play) your sketch, your canvas should now look like the image to the right.



15. Make sure you **add appropriate comments** throughout (at least 5 comments).

OK, here's a big sidebar you should read now:

VARIABLE TYPES IN PROCESSING

In the next few steps, we will begin working with variables.

Recall that variables can be described as containers used to store data. Unlike Scratch, Processing requires you to tell the computer your **data type**. Some common examples of types in processing are **int**, **float**, and **char**

- **int** (integer) is used for numeric data that does not include a decimal point
- **float** (floating point, a.k.a. decimal point) used for numeric data with a decimal point
- **char** (character) contains *one* typographical character such as a letter or symbol

In Scratch you declare that your program will be using a variable by clicking "Make a Variable" then typing the variable's name. The equivalent in Processing is to tell processing the **data type**, the **variable name**, then the **initial value** (optional). The general syntax used for **int** and **float** and most other types is:

```
type name = value;
```

The syntax for **char** is slightly different--the value must be contained by double quotes:

```
char var = "value"; //Note the use of quotes
```

Recall from Scratch that a **list** (a.k.a. an **array**) is a group of variables. This is also true in Processing. Arrays in general have the following syntax:

```
type[ ] name = {value1, value2, value3, … };
```

String is a special class in Processing that contains an array of characters (the char type). The basic syntax is similar to char:

```
String var = "This can be a sentence";
```

PITFALL: "String" is capitalized in Processing because it is technically a class and not a type. Don't worry about what this means now, but try to remember that "String" is capitalized.

VARIABLE SCOPE IN PROCESSING

In Scratch, you are asked if the variable should apply "to all sprites" or "to this sprite only" when you create it. For instance, a whole Scratch game might have just one "score" variable (choose "to all"), but four bouncing ball sprites might each have their own "velocity" variable (choose "to this sprite only"). In Processing, the equivalent idea is Global vs. Local **scope.** In the above example "score" is global and "velocity" is local. ***Where* you declare the variable** in your Processing sketch controls the **scope** – this affects where the variable can be used later.

Global variables contain data that you want to make available to the entire sketch (program). These variables need to be declared *outside of any blocks* or { }. It is good programming practice to put your globals at the top of your code.

Local variables are specific to one block ({ }) -- the one they are declared inside. If you declare a variable within a block then refer to it outside the block, you'll get an error.

16. In the text editor *above the Setup function* (not inside any function), **type** the following:

```
String str = "Go";
String str2= "Blue";
```

17. We need to select a font that will be used for the text. The first step is to call up the font class PFont as a global variable (check the PFont Reference page for more details). In your editor **type the following** *above the setup() function*

```
PFont MYFont; //Change the font name you use here
```

POINT OF CONFUSION: MyFont is the name you (the programmer) declare for the font. It does not have to match the actual name of the font in the operating system. For example, if you use "Helvetica" for important titles in your sketch, you might use "ImportantTitles" instead of "MYfont" in the code above.

18. Inside the setup() function, **type** the following:

```
MYFont = createFont("Georgia", 64, true);
```

Note since you changed the font name above, there should be a slight variation here with your code.

19. In the draw() function, we will type this text into our poster. **Type** the following code

```
fill(#00274C);
text(str, 210, 410);
text(str2, 180, 465);
```

It might be obvious, but the text() function takes on the following format – text(string name, x, y). **Run** your program now!

20. Oops! There seems to be something wrong with the text. **Take a screen shot and paste this into your lab report**.

21. Although we created a font and set its size above in Step 16-17, we need to tell the draw() function to use the font that we created in setup(). This is done using the textFont() function. Now in your draw function **type** the following

```
textFont(MYFont);
```

Make sure to type it above the text() commands you entered earlier. Rerun your code and take another screenshot of the result. **This will also be pasted into your lab report.**

RECAP: Writing text in Processing has four steps--we just covered them. First, you declare your own PFont to hold the font, then use createFont() to put a font from the operating system in there. Next, set the current font with textFont(). Finally, write some text with text().

22. We are almost done with our mascot. First let's create some scrolling text for our drawing. Recall in Scratch you could set a value for the x coordinate with the "Set x to _" block. We will be doing something similar here. **Type** the following command in the right place in your sketch that ***sets x as a global variable.***

```
int x;
```

23. Next let's create some text that will scroll across the screen. This will also be a global variable. **Type** the following command

```
String str3 = "This is Michigan!";
```

24. In the setup() function, **set the variable "x"** it to the value of a built-in variable, width.

```
x = width;
```

25. Now in the draw() function, **place the string** we just created at the following co-ordinates (x, 620). [If you are lost here review Step 17].

26. If you pressed Play (Run), you may have noticed that nothing seems to have happened. This is because the x coordinate is set at width so the text begins just off screen on the right-hand side. Let's fix that. In the setup() function, **change the x parameter** to width/2.

27. Now let's get the text to scroll. In the draw function **add** the statement:

```
x = x – 3;
```

Then press Play (Run).

EXTRA HELP: Confused? Remember that the draw() function works like the "Forever" block in Scratch, it is a loop that repeats while the program is running.

28. You can also **change the speed** of the scroll. Replace the last x = statement with this one which uses the Processing decrement operator (--):

```
x--;
```

What happens to the speed? **Enter your answer in your lab report now.**

TIP: Unlike Scratch, in Processing it is possible to draw objects that are entirely off-screen. Sometimes this is due to a mistake in your code, but in this case it is useful.

29. There is a problem with the scroll: once it reaches the left-side of the canvas it disappears completely and never returns! We don't want that to happen so let's write a conditional to **reposition the text** if it has moved out of sight. Type:

```
if (x <= 0) {
        x = width/2;
 }
```

Then **Play** (Run) the sketch.

Having worked with Scratch, we can figure out this conditional. "<=" means "less than or equal to". Since we are using the variable "x" to set the horizontal position (x-position) of the scrolling text "This is Michigan", and we know the left edge of the screen is at x=0, this conditional could be translated to mean:

> ***if*** *the horizontal position of "This is Michigan" is past the left edge of the screen*

The rest of the block (in between { }) would translate to:

> ***then*** *reset the horizontal position to where we started (width/2)*

However, this does not produce a very satisfying scroll effect. We need a way to tell if the right edge of "This is Michigan" goes offscreen, not the left edge. And it looks like the variable "x" corresponds to the left edge of "This is Michigan".

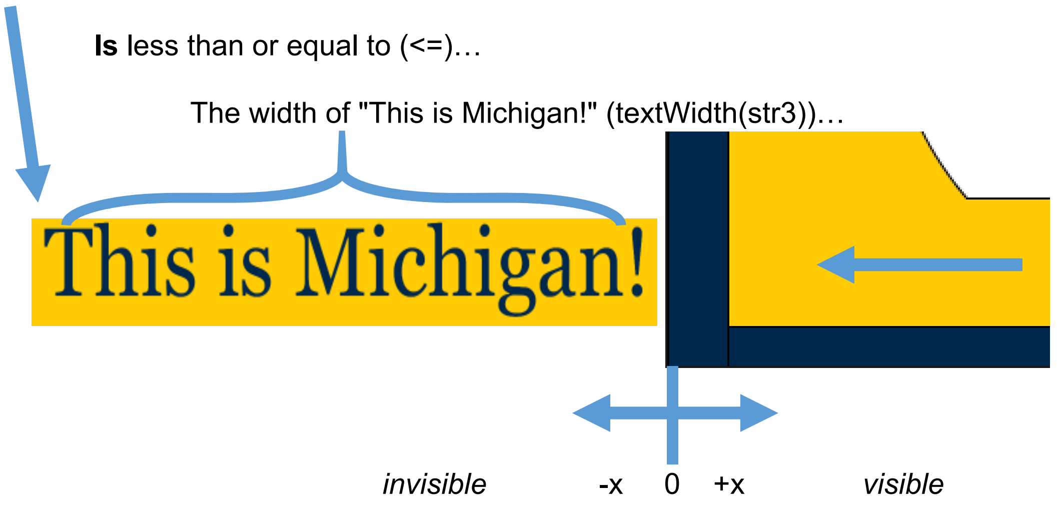
30. Let's try a different strategy. Modify the if conditional () clause to read:
```
if (x <= -textWidth(str3)) {
```
The textWidth() function calculates the width of a text string. It turns out that the width of "This is Michigan!" is 487.53125 pixels on the computer we're using to test this assignment, given the font we chose (yours will differ).

Since we now know "x" refers to the left edge, we therefore need to write a conditional that says: Is the left edge of this thing (x) 487.53125 pixels past the edge of the window (x = 0)? If so, that will mean the text is out of sight. This parenthetical if condition will do that. We could therefore translate the revised parenthetical () of this if statement to mean:

If the location of the left edge of "This is Michigan!" (x)…

Is less than or equal to (<=)…

The width of "This is Michigan!" (textWidth(str3))…



…but to make the above work we need to *add a negative sign* (-), because we want to detect when the text has passed the left edge of the canvas to become invisible to the user.

Finally, if the above conditional is true, we're telling Processing to move the text back to the mid-point of the canvas.

Question: How would you write this conditional if the text scrolled *right* to detect if the text scrolled off the *right side* of the canvas and became invisible? Write the answer to this question in your lab report now.

31. Right now we only have 1 set of the scrolling text. Let's make multiple copies of the text. We will use a conditional again. Below the last conditional in draw() { }, **add the additional code**:

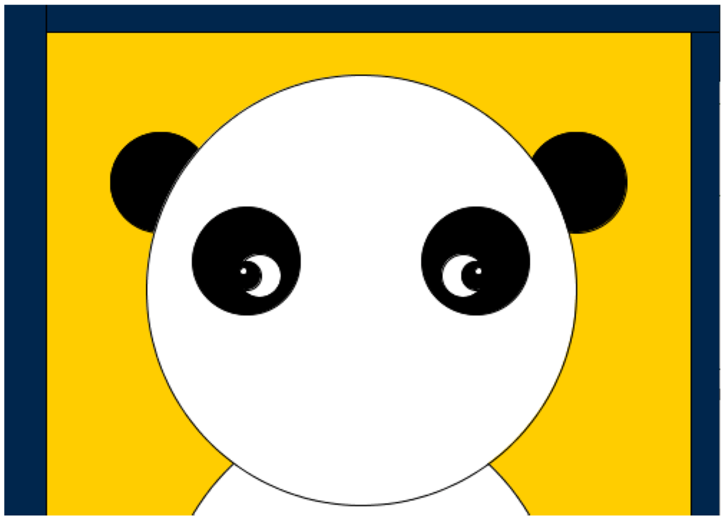
```
if (x < 0) {
        text(str3, x + textWidth(str3) + 100, 620);
}
```

This code says, if the left edge of "This is Michigan!" passes off the screen to the left (x < 0), then draw a new copy of the string to the right of it, with a 100 pixel gap in between them.

PRO TIP: Like with most programming concepts, there are multiple methods for creating scrolling software. Check out the Strings and Drawing Text tutorial for an alternative method that uses an array instead.

32. The Mascot is almost done. We have come to the end of the guided part of this portion of the tutorial. Now you get to choose your own adventure. **Make two additional changes** to the mascot
    - First, add either a nose or mouth. Adding both would count as one change. **(Hint: Using the curve() function will probably simplify your code tremendously).**
    - Second, change the if() conditionals and the use of the variable "x" to improve the scrolling text so that the two copies of str3 scroll past smoothly from the right edge to the left edge (without appearing or disappearing suddenly on-screen).

BONUS OPPORTUNITY: The eyes in our original mascot are fairly basic. Here is an example of some eyes with additional detail. See if you can replicate the eyes below using techniques you have learned. **To get credit for this bonus work, copy the code you used into your lab report.**



NOTE: In the guided part of this tutorial above we gave you the coordinates for each object you drew. You may have to do some mapping out in your head to get the coordinates right and may need to take a few educated guesses. If you were observant above, you will notice that often the x or y coordinates are similar. Make sure you do have some symmetry in your programming. In other words, we would expect to see a mouth or nose where one would be normally located.

33. You have almost reached the end of Part B of this tutorial. Now it's time to save your drawing. You will be saving both the code and directing Processing to save the actual drawing. In the editor, add the following code at the bottom (in a new block):

```
void keyPressed() {
        save("myUMuniqname_Mascot_file2.png");
}
```

Processing has some additional functions that are triggered by the mouse or keyboard. Adding the above void keyPressed() { } block of code is equivalent to adding the Scratch event "When any key pressed." It then uses the save() function and specifies an image name (Mascot) and the format of the image (png) to be saved.

PITFALL: You must type void keyPressed() ***outside*** of the draw() or setup() functions.

34. Run (play) the sketch, and press any key to save an image. After you stop the sketch, locate the saved image file in your Processing folder by using the pull down menu item: "Sketch" > "Show Sketch Folder". You will be submitting it as part of this assignment. This will be the 2nd file you submit for this assignment.

35. Now save your Processing code using "File" > "Save As…" with the filename ***myUMuniqname_sketch3***. This will be the third file you submit for this assignment. Saved sketch files end in the suffix ".pde" (Processing Development Environment).

PART C: Additional Processing Techniques

Overview: In this section you will gain some additional practice writing Processing Code while incorporating interactivity and movement. You will also learn how to use functions to simplify your code. Unlike Part B, there is less hand-holding here as the point is to get you thinking about the steps required. However, the exercises will be a series of short quick sketches.

Getting comfortable with the Basics.
The first 9 steps here should be familiar to you from Part B. This is your opportunity to practice what you learned. However, if you are feeling extremely lost here the code used here will be available on the Help Page. *Don't take this easy short-cut* unless it is absolutely necessary. Writing code on your own will help you gain programming skills faster.

1. Start a **new sketch** by clicking "File" > "New" in Processing

2. Create **two global float variables**. Name one *slow* and the other *fast*. Set the initial values for both to 0.

3. Create a **canvas** of size 400 x 400 pixels

4. Add a **draw()** function and change the **background color**. Note you can use hex color codes or rgb color codes to change the background color. (See the help page for this assignment for codes.)

5. Use one of the **shape** functions (ellipse, rect, quad, triangle) with starting coordinates (slow, 50). The other dimensions are up to you but keep your object fairly small relative to the canvas size. Use an appropriate fill() color for your shape.

6. Within draw(), **increase the value** of the variable *slow* to cause the object to move while the sketch is running.

7. Repeat step 5 to create a **second shape** using the coordinates (fast, 50)

8. Repeat step 6 but have this object **move forward faster** than the first shape.

9. Create two **if conditionals** that reset the position of your objects once they scroll to the end of the Canvas.

10. **Save** your sketch's code as ***myUMuniqname_sketch4***. (NOTE: You do not need to save the drawing). This will be 4th file you will submit for this assignment. Do not close this file you will be using it in the next few steps!

Using the Mouse and Keyboard

As we saw at the end of part B, Processing has additional functions that are triggered by mouse and keyboard events.

11. Create a new **keyPressed()** function by typing the following

```
void keyPressed() {

}
```

PITFALL: void keyPressed() { } must be *outside* of the draw() or setup() functions.

12. **Move the code** you typed in Step 6 inside the keyPressed function like this:

```
keyPressed() {
        Your code here from Step 6
}
```

13. **Run** your code. (Note that you will need to press down any key on the keyboard to have your shape move forward.) You might also have noticed that the shape moves very slowly. Let's **change the move speed** of the variable *slow* to an increment of 10 each time keyPressed() is called.

14. Save your file as ***myUMniqname_sketch5***. This will be the 5th file you submit for this assignment.

Now we will make a short sketch that uses the mouse to draw our shape.

15. Start with a **new sketch** with "File" > "New"

16. Create a **new canvas** setting size(600, 200) and set the background to 255. Set the **background** within the **setup**() function, like so:

```
void setup() {
        size(600, 200);
        background(255);
}
```

17. Initiate the **draw**() function and **type** the following code inside

```
fill(200, 0, 0);
ellipse(mouseX, mouseY, 33, 33);
fill(120);
ellipse(pmouseX, pmouseY, 21, 21);
line(mouseX, mouseY, pmouseX, pmouseY);
```

These functions are not new, but now we are using the position of the mouse (with the built-in variables: **mouseX**, **mouseY**) as our x and y coordinates to draw a circle. Processing's mouseX is the same as Scratch's "mouse x".

This code also uses another built-in Processing variable--the previous mouse position(**pmouse**)--to draw another circle. Finally, we create a line between the current mouse position, and the previous mouse position). **RUN** your program now (if you haven't done so to see how this works)

18. You may have noticed in the previous step that the drawing never ends as long as you move the mouse. What if you don't like the design that is developing? You can **reset the screen** using the mousePressed function. As a separate function outside the other blocks ({ }) type the following:

```
void mousePressed() {
        background(94);
}
```

Run the program again and make sure to click the mouse to see how this works.

19. Save your file as ***uniqname_sketch6***. This will be the 6th file you submit for this assignment.

You may have noticed at this point that we have placed **background**() in both the setup() and draw() blocks. These perform differently. When placed in setup() it sets the background for the canvas, but in draw() it continuously redraws the background color each time the draw function runs. It is important to think carefully about what you want to accomplish and where it would make the most logical sense to functions like background(), fill(), scale(), and the like.

Drawing Random Animated Shapes
One of the great things about Processing is it allows you to create random shapes that can be animated. Here we will learn some basic animation techniques.

20. Begin a **new** Processing sketch

21. **Create** a canvas of size 400 x 400

22. Begin a new **draw()** function and set the **background** to 0 and the **stroke** to 255. Although we haven't used stroke before the syntax is the same as that of background(). Look up the stroke() function for more information.

PITFALL: Note that the last step asked you to put the background and stroke functions within draw() and *not* setup().

23. Inside the draw function create a **float variable** called **xcord** and set its initial value to a random number between 0 and 400. Your code should be structured somewhat like this: (this is incomplete – fill in the xxx's)

```
xxx xxx = random(xxx);
```

REMAIN CALM: You will receive a warning that "xCord" is not used. That's because you haven't used it yet, but you will.

TIP: random() also accepts two values if you want to set both an upper and lower limit.

REVIEW: since xCord is inside draw(), that means it's a *Local* variable, not a *Global*.

24. **Create** a line with coordinates (xcord, 0) and (xcord, 400)

25. **Create** an ellipse with center (xcord, xcord) and with a width and height of 33.

26. **Run** your code. This should produce a number of flickering lines and circles on your canvas.

Question: As we noted above, the background and stroke functions were placed within draw() and not setup(). Why do you think this is? You may experiment with changing the location of these two functions to see what happens. Enter your response in the lab report, but be sure to replace them in the draw() function before saving your file.

27. Save the file as ***uniqname_sketch7***. This will be the 7th file you submit for this assignment.

Rotating Objects and Writing Functions

28. Create a **new Processing sketch**, then create a new float variable named r

29. Create a **canvas** of size 400 x 400 pixels.

30. In the **setup()** block, set the rectMode() to CENTER. This changes the default starting point for the rectangle from the top-left corner to the center.

31. We will now **create a new function** that will be used to draw rectangles of various sizes and will rotate them on their axis. In Scratch, new functions could be made by pressing "Make a block." In Processing, create a new function by specifying what it will return ("void" means nothing), naming it, then specifying what inputs it requires in parentheses:

```
void draw_rect(float x, float y, float rect_size, float rot) {
}
```

PITFALL: Place your new function outside draw(), setup(), or other functions.

32. Inside the draw_rect function's curly braces { } **type** the following code:

```
translate(x, y);  // Changes the origin of the canvas
rotate(rot);  // sets angle of rotation to the last coordinate of the rect
rect(0, 0, rect_size, rect_size);
resetMatrix();
```

For more about the translation and rotation functions, review the 2D Transformations tutorial

33. Now **create your draw()** function. In the draw function, set background() to 255, and the fill() to 0.

34. Still within draw(), **type** the following code:

```
draw_rect(100, 100, 80, r);
draw_rect(300, 100, 40, r * 0.3);
draw_rect(100, 300, 100, r * 0.6);
draw_rect(300, 300, 20, r * 1.2);
draw_rect(200, 200, 150, r * 2.3);
r = r + 0.02;
```

35. **Run** the code. You should see 5 rectangles rotating at various speeds.
36. **Save** your sketch as ***uniqname_sketch8***. This will be the 8th file you turn in for this assignment.

PRO TIP: Writing separate functions that you can call over and over again is a useful technique for simplifying your code. It means fewer lines of code, but it can also be useful for preventing errors and typos. In the example above we wanted to create 5 rectangles that rotate on their axis. Without the draw_rect function, the lines of code from translate(x, y) to resetMatrix(); would be needed for each rectangle we create.

Using Loops to Automate your Sketch

One key to writing good code is to find ways to simplify. In the following exercise, you will be introduced to while(), which is a loop that can automate several steps. In Scratch, a similar conditional to the while loop is "Repeat until". In this exercise, the goal is to draw a number of circles on the canvas. However, we will let Processing do most of the work with a while loop. The while loop is actually a conditional which states, "Do x as long as a given condition remains true."

37. Start a **new**, blank sketch

38. Let's create a global color palette to use with our while loop. Type the following **array**:
```
color[] design = { #f0f0f5, #ffccff, #cc9900, #99ccff, #0033cc, #00cc66, #ffcc00};
```

39. We need to create a **global variable** that will be used to set the condition for the while() function. Type the following code now.
```
int circ = 0;
```

40. Now let's **create our canvas**. Use setup() and make it size(600, 300) and add the following code:
```
background(design[0]); // set background to 1st color in array
```

41. Create the draw() function and type the following inside. This is the basic syntax for the while loop:
```
while(circ < 12) {
}
```

42. The rest of our code goes inside the while loop's block: a.k.a. the { }s .
```
fill(design[int(random(1, 6))]); // randomly choose a color from array
float x = random(width);
float y = random(height);
float d = random(30, 300);
ellipse(x, y, d, d);
circ = circ + 1;  // increments circ by 1.
```

PRO TIP: Although while loops can be very useful, if they are not designed carefully they could lead to an endless loop which will halt your program. Be sure to check the condition inside the while() parentheses to be sure it will be triggered at the right time.

43. **Run** your program to make sure it is running correctly. You should have 11 circles randomly drawn on your canvas. **Save** the file as ***uniqname_Sketch9***. This will be the 9th file you turn in for this assignment.

PART D: Working with Data

Objective: In this section, we will manipulate text and learn how to acquire external data.

Manipulating Strings

REVIEW: Recall that letters can be represented using the **char** data type or **String** class. char is used for storing a single letter (i.e. "a", "b") and String is used to store a group or an **array** of characters (i.e. "abc", "hello").

Also recall that **String** and **char** work like **int** and **float**, storing the values you want the program to remember for later use, like so: String greeting = "hello"; or char c = "Z";

Now let's begin to some basic manipulation of Strings.

1. Open a **new sketch** in Processing

2. **Declare** the following global variable in the editor
   String message = "March madness";

3. Now let's make a new global string variable that is entirely uppercase using the toUpperCase() **method** without re-typing the text. Type:
   String uppercase = message.toUpperCase();
   We now have a new string that holds the original message in uppercase form.

WTH is a METHOD?! A method() is just a named collection of statements in a program, similar to a function(), except that methods apply to particular types of data. Processing comes with a lot of methods already defined for you. (In fact you've already been using them.) When we say a String variable has "methods" we mean that there are functions that can be invoked to act on a particular kind of object (a string). Methods are written ***after*** the object they act upon, separated by a dot, as in step 3 above. For example, a method that applies to SomeObject might be written like this:
SomeObject.MethodThatWorksForThatObject(); // note the dot ('.') in there!
Methods that work for String objects are listed in the Processing reference under String. Scroll down past "parameters" in the Processing help and look for "methods."

4. To see if it worked, we can print this string to the console. Create the setup() function and add this **println()** command to it:
   println(uppercase);

SUPER USEFUL: println() and the console are extremely useful to check to be sure your code is doing what you think it is doing. For instance, you can add println() statements to check how your variables and arrays are changing as your program runs. This is similar to using the checkbox in Scratch to make your variables visible.

Suppose however that we want to extract different characters from the string. Recall that strings are an **array** of characters. Each character in the string is **indexed** by a number, **starting from 0**. In the String above, "March madness" has 13 characters, indexed from 0 to 12. This also explains why in Part C, Step 40 we used background(design[0]); to select the first color in our array – the first color is "0".

USELESS INFORMATION: There are a lot of bad programming jokes that hinge on the fact that programmers like to start counting things from "0".

PITFALL: Whitespace and punctuation marks are included in the strings, so a string like "Go blue!" would have 8 characters, indexed from 0 to 7.

To extract different characters randomly from the string we would need to:

a) Figure out how long the string is (the last index of the string)
b) Choose a random number that can range from 0 to the string's last index
c) Print the character at that index number
d) Repeat the above five times

Translating that to Processing, we first create a for() { } loop that will allow the program to repeat several steps a set number of times. The above steps are implemented "from the inside out" which is often a good way to think about writing loops.

5. Type the following into the text editor

```
for (int i = 0; i < 5; i++) {
        int RandomCharacter = int(random(0, uppercase.length()-1));
        print(RandomCharacter);
}
```

What do you notice in the console? Enter your answer in your lab report now.

********* EXTEND YOUR KNOWLEDGE*********

The syntax of the for() { } loop can be tricky if you are new to programming. Check the documentation!

'int i = 0' initializes the variable "i" which we will use to keep track of how many times our loop has been executed. Since programmers count from 0, we'll start at 0. 'i < 5' tells the program the condition under which the loop should be executed. This will stop the loop as soon as I = 5 (or greater). Finally, 'i++' increments i by 1 each time the loop executes. To *increment* a variable means to add one to it.

Next, our loop begins in the block { }. We initialize another int variable (RandomCharacter), which we use to hold the randomly-generated array index for the character to print… in other words, the character's position in the String "uppercase."

Recall that the **random()** function (see Part C, Step 23) has an lower and upper limit. In this case we have asked Processing to randomly select any character in the string, with 0 obviously being the first character. However, notice we did not simply count the number of characters in our string (which in this case is 12). Rather we asked Processing to calculate that for us. Also, because the numbers are automatically **floating numbers** (decimals), we needed to wrap our random() function in an **int** to convert the type to integer.

In Part B, we used the textWidth() function to calculate the length of our text. Here we used the length() function. The length() function, when used as above, will return the total number of characters in a string. But in a string the characters are indexed starting from 0, so we subtract 1 from that number.

6. We still haven't told the program to print the **specific characters** stored by those indexes. The **charAt()** method helps us do this. (The charAt() method can be applied to Strings objects.) Upon passing a number to charAt(), the program will return the character of that index.

   In your **for()** loop, add the following line of code:
   println(uppercase.charAt(i) + "!");

7. Run the program. You should have some letters now displayed in the console.

8. Save your file as ***uniqname_sketch10***. This will be the 10th file you turn in for this assignment

Visualizing Data with Processing.

9. Open a new sketch in Processing

10. Now, **download** the JSON file cats.json and add it to your Processing sketch's canvas. You add the file to your canvas by going to **Sketch → Add File**. This means the file cats.json is added to your "data" folder for the current sketch. If your sketch is exported, these files in "data" will be exported too. This is useful if your program needs to work with external data files when it is installed on someone else's computer.

    Double-check: Make sure the file is in the **data folder** of your sketch! You can check by going to **Sketch → Show Sketch Folder**.

11. Declare a *global* JSONArray (a particular kind of array) for JSON objects by typing the following code in the PDE:
    ```
    JSONArray cats;
    ```

12. **Create** a new canvas of size(600, 400) with a white background (255)

13. Next we will load our JSON file using the loadJSONArray function. This function takes the name of a JSON file as its argument, opens the external file, reads it, and then stores the JSON data in an appropriate object in Processing for use by your code. We're going to store these data in the **cats** JSON array we declared earlier. **Type** the following code in the setup function:
    ```
    cats = loadJSONArray("cats.json");
    ```

14. Before we get to visualizing our data, let's take a look into what the JSON file holds. Try opening the **cats.json** file in your text editor, what do you see? It looks like a small spreadsheet that someone exploded with extra punctuation. In fact JSON stands for JavaScript Object Notation. JSON is a format that uses a text file to contain data objects used by computer programs. Unlike a spreadsheet, each JSON object might have another JSON object inside of it… and that one might have another JSON object inside of that. Now type the following *into a function where it will run only once*:
    ```
    JSONObject cat = cats.getJSONObject(0);
    //This tells Processing to retrieve the first object in the JSONArray 'cats' and
    ```

// to store it in the JSONObject 'cat'

15. Next let's extract some information from the first JSON object in the array. Although the file contains several data elements for each object, we only want the name, breed and age of the cat. **Type** the following after the statement you added in the last step:
    ```
    String name = cat.getString("name");
    String breed = cat.getString("breed");
    int age = cat.getInt("age");
    ```

16. Use the println() function to print this information to the console
    ```
    println(name + ": " + breed + ", " + age);
    ```
    Notice the use of the + signs. This is a useful way to join strings called "concatenation."

The next part of this exercise will require you to come up with your own code. Follow the instructions carefully so you don't miss any steps. If you get lost some help will be provided in the help resources for this assignment.

17. So far so good. Now we would like to draw some of this data to our canvas. We will draw a simple chart that visualizes the age of the cats. **Create** a draw() function and set the stroke fill color to 153 like so:
    ```
    stroke(153);
    ```

18. At the top of our canvas, let's **add an array of color values**. You can use the same array as before (See Part C, Step 38 or create a custom array of your choice).

19. We will now create a **for loop** that automates the reading of the JSON file. In the previous step (Step 14-15), we went over printing information from a **single** JSON object. But our file is an **array** of JSON objects (in other words, a list of cats), we want to read in data for every cat and use it to create the chart. We could simply retype the code in Steps 14 and 15 for each object in the array (0, 1, 2, etc), but what if the array contained 150 cats? Or an unknown number of cats. We want to **avoid** using repetitive code when there is a faster way: a **for() { } loop**. Complete the loop and **type** it into the draw() function:
    ```
    for (int i = 0; i < cats.size(); i++) // Do not forget to include the curly brackets
    ```

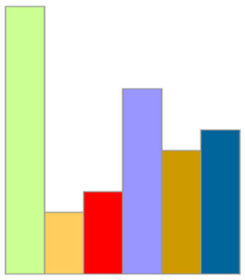
    You should already be familiar with the first part of the for loop (see Step D5). This loop tells the computer to start at 0, then count to the number of items in the JSONArray, one at a time. The **cats.size()** command uses the size() method on the array "cats" to find out the number of items (also called the "length") of the array, so the program knows when to stop. Lastly, **i++** increases the counter variable "i" by one after each time the loop has been executed – hopefully after each cat in our array of cats.

20. Now within the **for () loop's block { }**, declare and name a new local JSONObject to hold one cat. Get a JSONObject from the array "cats" and store it in your new local JSONObject. This is very similar to what we typed in D14, but instead of retrieving the first cat in the array by using array index 0 (which would always return the first item), you should have use a variable so that your JSONObject will receive the appropriate cat when the for () loop is executed (a different cat each time the loop is executed, in other words).

21. Next, still within the for() loop, declare a new local variable of the appropriate type to store the age of each cat. Fill the variable with the age of the cat from the JSONObject you just created. This is similar to the code we typed in D15.

22. Add a **statement** within the for() loop to set the **fill()** color so that the subsequent shapes that are drawn by the code within the for() loop will have a different color each time the loop is executed, with colors selected from our color array initialized earlier**.** (**NOTE:** This is a little tricky. It bears some similarity to the code we typed in Part C, Step 42 **BUT** here we simply need to select each color from the array in order. Your fill code should include [i].)

23. Finally, let's add a statement to **create** the bars in our bar chart to represent each age by drawing some rectangles. We're aiming for something like this as our result:



IMPORTANT PLEASE READ: This is probably **the most complicated** code in the for () loop so let's think this through. Recall that most rectangles begin with an origin point (x, y) and then a width which moves towards the right of the canvas, and a height that moves downwards.

Here we need to draw a rectangle where the height moves in the opposite direction, because in a bar chart each bar varies at the top and not the bottom. The origin point for your rectangle should be close to the bottom of the Canvas and the height parameter in your rect() statement should have a negative sign, so that **the bars will grow up as the age values go up**.

Our bar chart represents cat ages. You'll also need **the height of the rectangle to represent the age of the cat**, so age needs to be represented in your rectangle's height parameter.

The for loop is going to draw several rectangles, but if all of them have the same x parameter we'll **draw them on top of each other. That would be bad!** So be sure to set the rectangle's x coordinate with a variable in such a way that it moves over each time the loop is run.

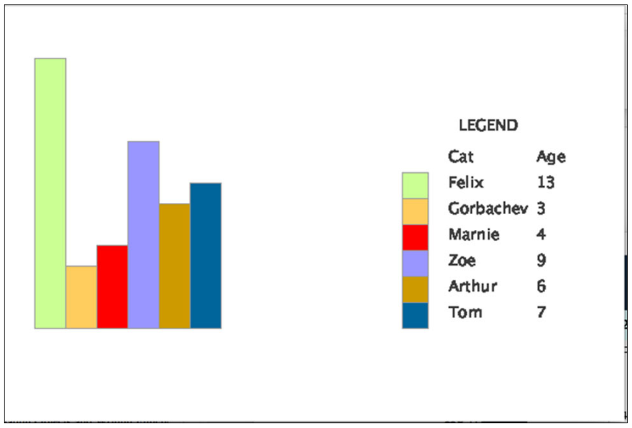
Finally, because there are only 6 objects in the JSON file, your array moves from 0 to 5. Graphically this will produce an unreadable bar graph if each bar is 1 pixel wide. So you will probably need to multiply a variable in the x value by something so that **the entire graph fits the canvas.**

HINT: Stuck? The final statement could look something like this pseudo-code. We've spaced it out for readability:

rect (**starting-x-position** + i * *some number to make each bar move over to the right*),
  starting-y-position, \
  the width in pixels of each bar,
  *the negative sign "age" object * some number to make the bars taller*);

If you have completed all the steps (17-23) above correctly, you should now have a chart that looks like the one above.

24. Right now our chart doesn't give us a lot of useful information so let's create a legend which we can use to tell what each bar stands for. Let's aim for something like this:



In the draw() function **outside the for loop**, type the following (in the interest of time we will use the default Processing text but you need to set the text size to 15):

```
//Set text size to 15. Use reference page for assistance
text("LEGEND", 440, 120);
text("Cat" 430, 150);
text("Age", 515, 150);
```

It is OK if your position parameters vary. These values are just examples.

25. In the for loop, let's also declare a local variable of the appropriate type to contain each cat's name from the JSONObject (similar to the code we typed in Step D14-15).

26. Now let's create some rectangles for the legend. In the for loop type something like the following (your rect positions may vary, this is just an example):

```
rect(385, i*25+160, 25, 25);
```

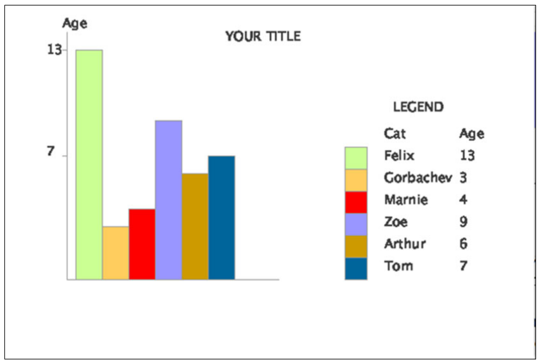
27. Now let's list each cat's name. First we want the color to be black so change the **fill to black.** Then **type** code like the following in the for loop (your positions may vary):

```
text(name, 430, i*25 + 175);
```

28. Now please repeat step 26 and 27 but for each cat's age. If you have completed this step correctly, your chart should now look like the last bar chart picture shown (not drawn to scale).

29. Our chart doesn't have a title. Use the knowledge you've acquired so far to **create** a title for the chart.

30. **Create** an X and Y axis for the chart similar to the following image. There are a number of steps to complete here:
    - Label the y-axis ("Age")
    - Create two small hashmarks on the y-axis that correspond to the age of the cat (i.e. the hashmarks should be at the same y-coordinate as the bar)
    - Label these with the age of the corresponding cat.
    - If you have completed these steps correctly, your final chart should look something like this.



31. Save your chart as a png file with the file name as follows: ***myUMuniqname_chart11.png*** (Review Part B, Step 33 if you can't remember how to do this. This will be the 11th file you submit for this assignment.

32. Now save your sketch as ***uniqname_sketch12***. This will be the 12th file you submit for this assignment.

BONUS WORK

Lab assignment 4 is over, but if you are interested in learning more about Processing (or programming), or the assignment was too easy for you, you can receive extra credit for bonus work. For this assignment, there are a few options for bonus work. You can decide how to demonstrate you have completed the bonus work.

For this assignment each bonus work task can receive up to the amount indicated of extra credit on this assignment, up to a maximum of +40%. Here are the options:

- **Fixing the Eyes of the Mascot:** Complete the Bonus Opportunity in (B32) for these points (+5%)
- **Improve the bar chart from Part D:** Work on the bar chart from Part D to make it look much nicer, or to present additional data from the JSON file. (+2.5% to 7.5%)
- **Amazing Visualization, Amazing Effort, Amazing Code**. If the effort you put into the visualization impresses the teaching staff, or if the visualization itself or programming impresses the teaching staff, you may receive extra credit. (+% varies)
- **Interactive Visualization**. Instead of a basic bar chart in Part D, make a visualization of some kind that changes based on the user's behavior. (This could be an interactive bar chart.) Ideally this would be useful. To put this in other words: clicking on it, moving the mouse, or pressing a key should change the visualization in a helpful way. A common example of this is to have sections that expand or collapse, or are shown/hidden. (up to +10% per significantly different interactive feature)
- **Different Visualization**. In addition to the bar chart in Part D, make a different visualization. This can be combined with the previous bonus item and/or the following bonus item. The visualization must contain for () loops and draw data from an external file. (+15%).
- **New Data Source**. Most social media sites allow you to export your data in JSON or CSV format. In addition to the JSON about cats, use a different source of data e.g., about your Instagram post history or some other JSON or CSV file that you obtain on the Web. Note that this may also ***require*** a different visualization (the previous bonus item) because the data will be different (up to +25%)
  - Example: This article explains how to download a JSON file of your Instagram posts, so that you can make a bar graph about that: https://lifehacker.com/whats-in-your-instagram-data-dump-and-how-to-get-it-1825777655
- **Real-Time Data Source**. Instead of using a JSON file saved with your project, figure out how ask your Processing program to download the data for you when you run it. (+5% for a simple download, but up to +40% if you use an API -- this may be challenging and require the use of additional libraries.)