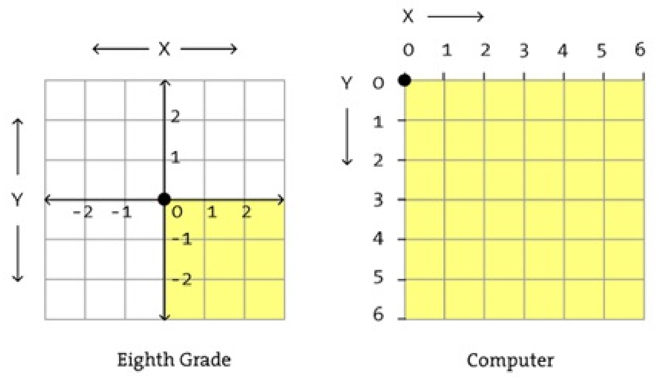
Use Processing to Build a Talking Robot Face

by Siyan Zhou, July 15, 2017

This tutorial introduce how to use Processing to build an animated and interactive talking robot face on your computer screen. It also covers some computer programming basics along the way. We assume you already have **Processing 3.0** installed on your computer. If not, you can read the tutorial by Casey Reas and Ben Fry to get started.

Coordinate System

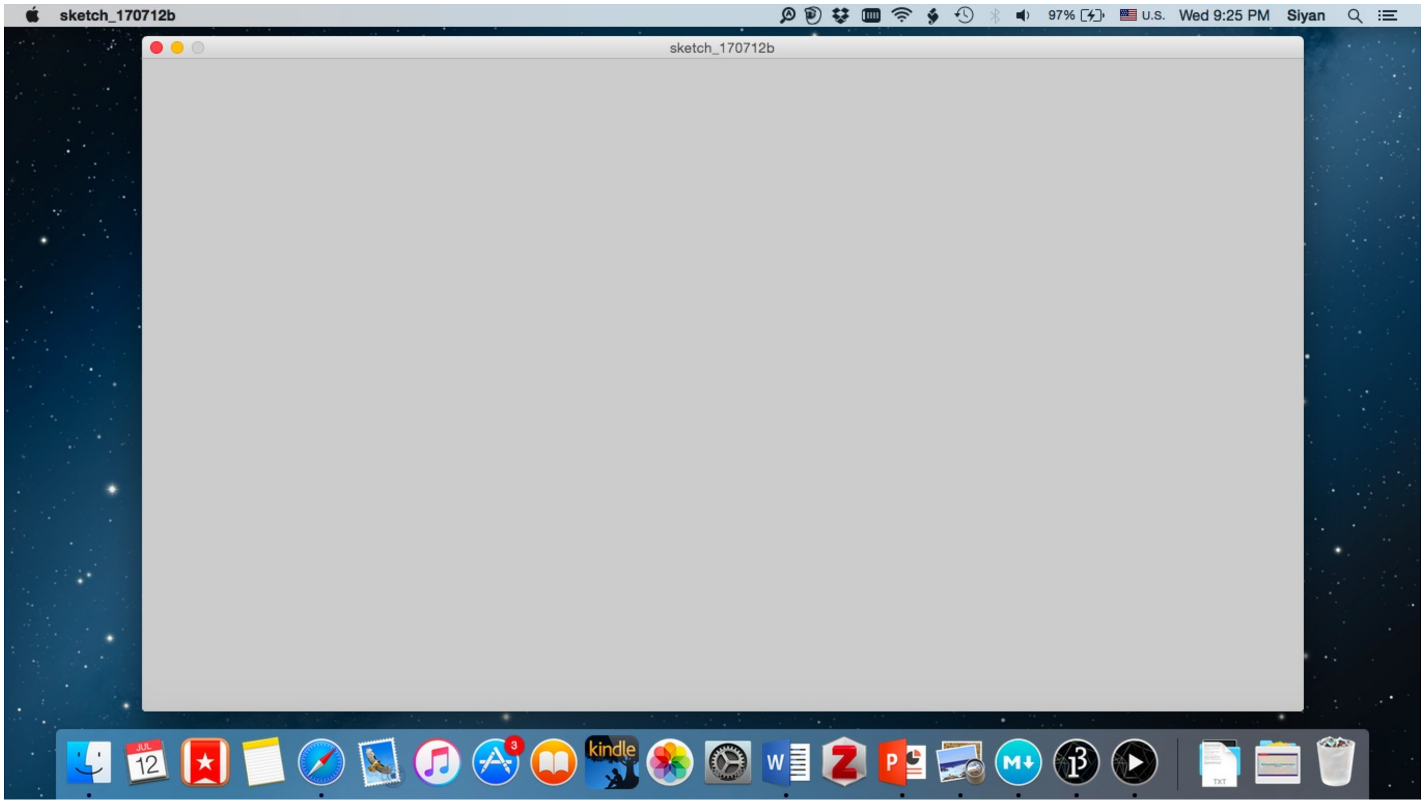
Before we begin programming with Processing, we must understand that the computer screen is like a sheet of graph paper. That is, each pixel of the screen is a coordinate - two numbers, an "x" (horizontal) and a "y" (vertical) - that determines the location of a point in space. Using the Cartesian coordinate system you learned in eighth grade, (0,0) can be found at the top left with both positive x-axis pointing to the right and positive y-axis pointing down. And it is our job to specify what shapes and colors should appear at these pixel coordinates.



But before you can draw anything on the screen, you need to tell the computer the size of a window, where you can define coordinates. You can type the size function in the Processing editor. A function is the instruction you want the computer to follow and it is written in a way the computer can read. Here, an size function example will look like this:

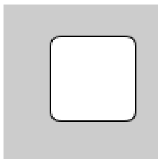
```
size(1120,630);
```

If you run this code in the Processing editor, an empty display window with 1120 pixels in width and 630 pixels in height will appear on the screen:



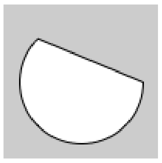
Simple Shapes

As you can see at the beginning of the tutorial, we need to instruct the computer to draw a big rectangle for the face outline, three arcs for the eyelids and lips, and then four ellipses for the eyes. Some shape functions will look like this:



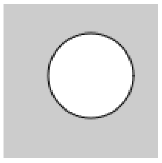
```
rect(x, y, width, height, radii);
```

- "x" is the x-coordinate for the top left corner of the rectangle
- "y" is the y-coordinate for the top left corner of the rectangle
- "width" is the width of the rectangle
- "height" is the height of the rectangle
- "radii" the radii for all four corners



```
arc(x, y, width, height, start, stop);
```

- "x" is the x-coordinate for the centerpoint of the arc's ellipse
- "y" is the y-coordinate for the centerpoint of the arc's ellipse
- "width" is the width of the arc's ellipse
- "height" is the height of the arc's ellipse
- "start" is the angle to start the arc, specified in radians
- "stop" is the angle to stop the arc, specified in radians

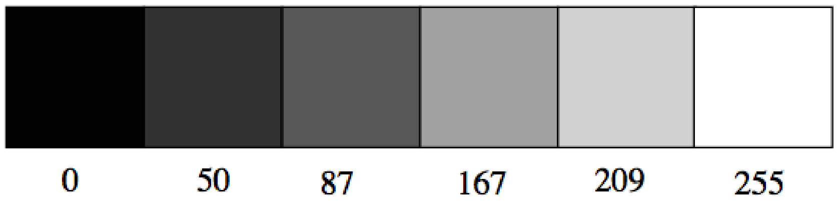


```
ellipse(x, y, width, height);
```

- "x" is the x-coordinate for the centerpoint of the ellipse
- "y" is the y-coordinate for the centerpoint of the ellipse
- "width" is the width of the ellipse
- "height" is the height of the ellipse

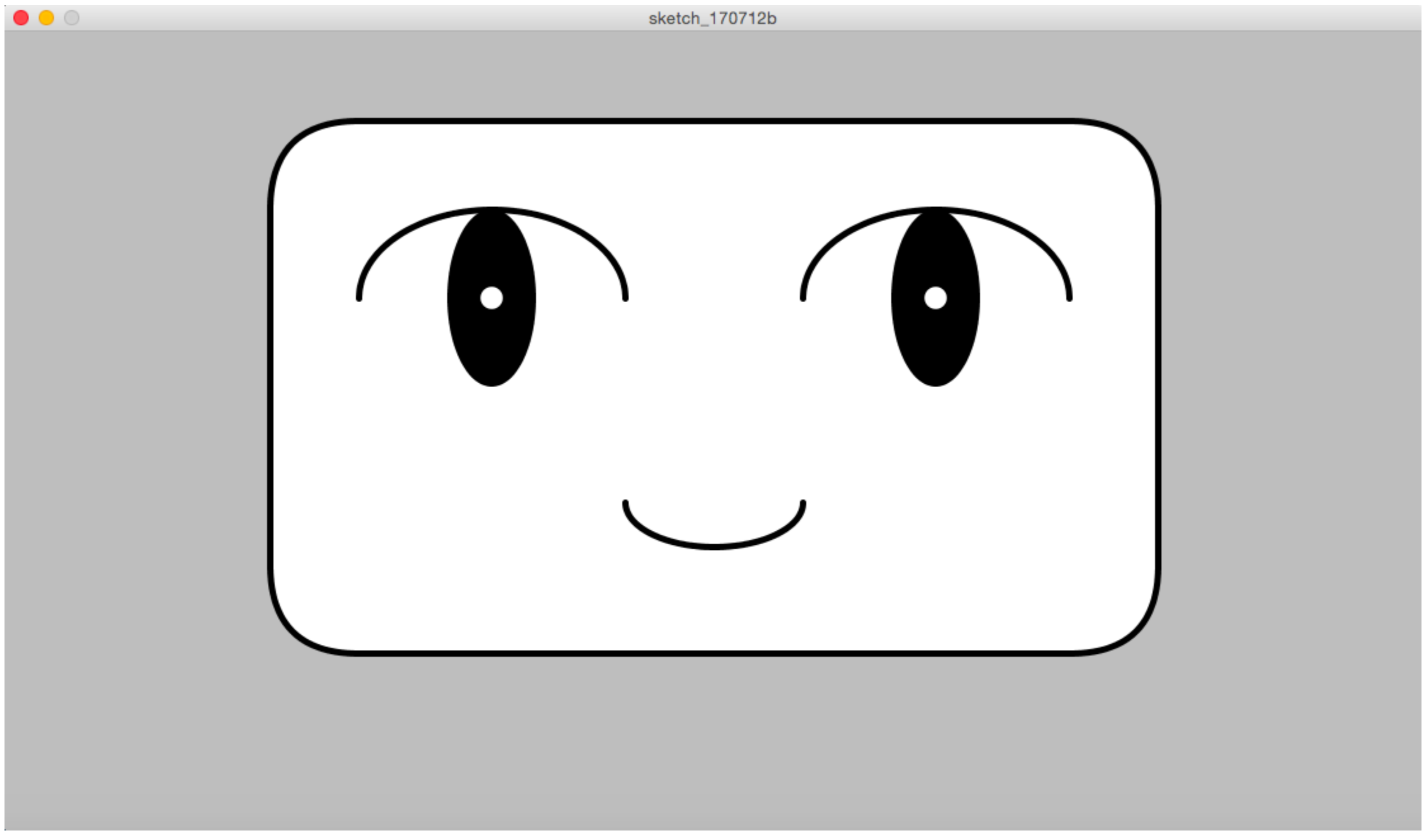
Color

In the digital world, color is defined as a range of numbers. In our robot face example, we only need to discuss about the simplest case: black & white or grayscale. 0 means black, 255 means white. In between, every other number from 50, 87, 162, 209, and so on is a shade of gray ranging from black to white.



By adding the `stroke()` and `fill()` functions before something is drawn, we can set the color of any given shape. There is also the function `background()`, which sets a background color for the window. Here is example code to draw the robot face (anything written after `//` in a line is a comment):

```
size(1120,630);                      //set the size of the window
background(190);                     //Set the background to gray

stroke(0);                           //Set the stroke to black
strokeWeight(5);                     //Set the thickness of the stroke
fill(255);                           //Set the interior the following shape to white
rect(210, 70, 700, 420, 70);         //draw the face outline

strokeWeight(5);
arc(385, 210, 210, 140, PI, 2*PI);   //draw the right eyelid
arc(735, 210, 210, 140, PI, 2*PI);   //draw the left eyelid

fill(0);                             //Set the interior the following shape to white
ellipse(385,210,70,140);             //draw the right big eyeball
ellipse(735,210,70,140);             //draw the left big eyeball

fill(255);
ellipse(385,210,17.5,17.5);          //draw the right small eyeball
ellipse(735,210,17.5,17.5);          //draw the left small eyeball


fill(255);
stroke(0);
strokeWeight(5);
arc(560,371,140,70,0 ,PI);           //draw the mouth
```


sketch_170712b

Flow(`setup()` and `draw()`)

Even though we are able to tell the computer to draw a robot face with simple shapes and colors, they are all static images on the screen. In order to make an animated face, we need to understand the concept of the ***control flow.*** As you can see in the code example for the robot face above, each line of code is generally executed from top to bottom. Control flow structures, however, can allow you change execution order and determine what section of code is run in a program at a given time. You can regulate the flow of your program's execution with control statements and loops. Here is an example of a control flow statement:

```
void setup(){
  //any line of code within these curly brackets only happen once.
}

void draw(){
  //any line of code within these curly brackets is executed
  //continuously each frame until the program is stopped.
}
```

Variables

One of the interactive features is that the robot's eyes track the mouse movement, so we need to vary the eye's location according to the mouse position. We need some mechanism for dynamically storing the mouse's location - **variables.**

Built-in variables

`mouseX` and `mouseY` are built-in variables in Processing 3.0. There are variables that stand for the X and Y locations of the mouse. `width` and `height` are also Processing built-in variables which store the the current value for the width and height of the window. Built-in variable `frameCount` contains the number of frames that have been displayed since the program started.

User-defined variables

We can also make up our own variables in three steps:

1. Declare a variable by defining the variable data type and giving a name, and then followed by `;`. What are some possible types? The common data types are `int` and `float`. `int` means integer or a whole number such as 3, -10, 209, etc. `float` means a decimal or floating-point number such as 2.3323093 and -100.334343. A name could be anything, but usually works with what you're using it for. Here is an example of declaring a variable:

   ```
   float X;
   float Y;
   ```

2. Initialize the variable. We give the variable an initial value, which will look like this:

   ```
   X = width/16;
   Y = height/9;
   ```

3. Use the variable in drawing the small eyeball of the robot face:

   ```
   ellipse(5.5*X, 3*Y, X/4, Y/4);
   ```

   Note: Variable names are case sensitive! This means that `variable` is NOT the same thing as `VARIABLE`!

Boolean variables

There is a special data type called a `boolean`. Unlike `int` or `float`, `boolean`only has two possible values: `true` or `false`.

```
boolean apple = true;
boolean banana = false;
```

Conditionals

As mentioned in the Flow section, you can regulate the flow of your program's execution with **control statements**. One type of control statements is conditional statement.

If, else if, else

There are `if`, `else if` and `else` statements, which do exactly what like they sound like they do:

```
If (mouseX > 100) {
  rect(210, 70, 700, 420, 70);
} else if(mouseX < 50) {
  arc(735, 210, 210, 140, PI, 2*PI);
} else {
  ellipse(385,210,70,140);
}
```

In this example, if the X position of the mouse is greater than 100 pixels, draw a rectangle. Otherwise, if X position of the mouse is less than 50 pixels, draw an arc. Otherwise, draw an ellipse.

Boolean expression

In the example above, all `if` statements are boolean expressions. A boolean expression evaluates only to `true` or `false`. There are many ways we can create a boolean expression such as using a boolean variable, but the simplest and useful way for us to start with is to use relational operators and logical operators.

Here are some examples for relational operators:

`>` greater than

`<` less than

`==` equal to

Here are some examples for logical operators:

`&&` and

`||` or

```
if(mouseX>100) || (mouseX<50) {
  rect(210, 70, 700, 420, 70);
}
```

This means if X position of the mouse is either greater than 100 pixels or less than 50 pixels, then Processing will draw a rectangle.

Animated Robot face

With conditional statements, we now have the ability to add some logic to the program and let the program take a path. We are ready to make an animated robot face now. But, before we can do that, we need to learn the concept of modulo and the `map()` function.

Modulo

Modulo calculates the remainder when one number is divided by another. For example, when 52.1 is divided by 10, the divisor (10) goes into the dividend (52.1) five times (5 * 10 = 50), and there is a remainder of 2.1 (52.1 - 50 = 2.1). Thus, 52.1 % 10 produces 2.1. In Processing, `%` stands for the modulo operator.

Map()

`map()` is a calculation function allows you to take any range and map a value inside that range to a new value in any other range. The `map()` calls five arguments and it looks like this:

```
map(value, start1, stop1, start2, stop2)
```

- `value` is the incoming value to be converted
- `start1` is the lower bound of the value's current range
- `stop1` is the upper bound of the value's current range
- `start2` is the lower bound of the value's target range
- `stop2` is the upper bound of the value's target range

Now we can take what we have learned to make the robot's eyes blink and track the mouse movement:

```
float X;                                                 //create a variable X for better documentation
float Y;

void setup(){
  size(1120,630);
  X=width/16;
  Y=height/9;
}

void draw(){
  background(190);

  stroke(1);
  strokeWeight(5);
  fill(255);
  rect(3*X, Y, 10*X, 6*Y, X);                            //draw a face outline

  if(frameCount % 350>0 && frameCount % 350 <25) {         //if eyes open every 350 frames and stay open for 25 frames, draw eyeli
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, 0,PI);                     //draw right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, 0,PI);                    //draw left eyelid

  }else {                                                //if not draw eyes open with moving eyeballs
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                 //draw right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                //draw left eyelid

    fill(0);
    noStroke();
    float eyeballA=map(mouseX,0,width,5*X,6*X);          //The X position of big eyeballs vary with the X position
    ellipse(eyeballA,3*Y,X,1.9*Y);                       //draw right big eyeball
    ellipse(eyeballA+5*X,3*Y,X,1.9*Y);                   //draw left big eyeball

    fill(255);
    noStroke();
    float eyeballBX =map(mouseX,0,width,4.9*X,6.1*X);      //The X position of small eyeballs vary with the X position
    float eyeballBY =map(mouseY,0,height,2.25*Y,3.75*Y);   //The Y position of small eyeballs vary with the Y position
    ellipse(eyeballBX,eyeballBY,X/4,Y/4);                  //draw right small eyeball
    ellipse(eyeballBX+5*X,eyeballBY,X/4,Y/4);              //draw left small eyeball
  }

  fill(255);                                               //draw a mouth
  stroke(1);
  strokeWeight(5);
  arc(8*X,5.3*Y,2*X,Y,0 ,PI);
}
```

Functions

We have been learning all this stuff about programming and it has been great, but now comes the time where we need to start looking at of organizing our code to make it more scalable for the future. Right now we are just writing some variables up at the top and putting some suff in `void setup(){}` and `void draw(){}`. It turns out that the `void draw(){}` is getting bigger and bigger and is looking pretty messy. We need to realize that there are some blocks of the code that draw the eyes, and there are some parts of the code that draw the mouth and face outline. You can actually start organizing your code by creating your own functions.

In fact, you are probably already pretty familiar with the concept of functions. When you write the code `background(190)`, you are calling a function. Someone created this fuction and then named it `background`. This person defined what `background()` does, and what codes should be executed when you call this function.

The way the function is defined is by giving it a return type and then we have to give it the name of the function. Then we give some amount of arguments and then open and closed curly brackets. Inside the curly brackets, we want to put codes that you want to execute for the function.

We can now create functions for the robot face, eyes blinking and mouth. The code will look more organized:

```
float X;                                                    //create a variable X for better documentation
float Y;

void setup(){
  size(1120,630);
  X=width/16;
  Y=height/9;
}

void draw(){
  background(190);
  faceOutline();
  eyesBlink();
  mouth();
}

void faceOutline(){                                         //draw a face outline
  stroke(1);
  strokeWeight(5);
  fill(255);
  rect(3*X, Y, 10*X, 6*Y, X);
}

void eyesBlink(){
  if(frameCount % 350>0 && frameCount % 350 <25) {          //if eyes open every 350 frames and stay open for 25 frames, draw eyeli
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, 0,PI);                        //draw right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, 0,PI);                       //draw left eyelid

  }else {                                                   //if not draw eyes open with moving eyeballs
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                    //draw right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                   //draw left eyelid

    fill(0);
    noStroke();
    float eyeballA=map(mouseX,0,width,5*X,6*X);             //The X position of big eyeballs vary with the X position
    ellipse(eyeballA,3*Y,X,1.9*Y);                          //draw right big eyeball
    ellipse(eyeballA+5*X,3*Y,X,1.9*Y);                      //draw left big eyeball

    fill(255);
    noStroke();
    float eyeballBX =map(mouseX,0,width,4.9*X,6.1*X);       //The X position of small eyeballs vary with the X position
    float eyeballBY =map(mouseY,0,height,2.25*Y,3.75*Y);    //The Y position of small eyeballs vary with the Y position
    ellipse(eyeballBX,eyeballBY,X/4,Y/4);                   //draw right small eyeball
    ellipse(eyeballBX+5*X,eyeballBY,X/4,Y/4);               //draw left small eyeball
  }
}
```

```
void mouth(){                                               //draw a mouth
  fill(255);
  stroke(1);
  strokeWeight(5);
  arc(8*X,5.3*Y,2*X,Y,0 ,PI);
}
```

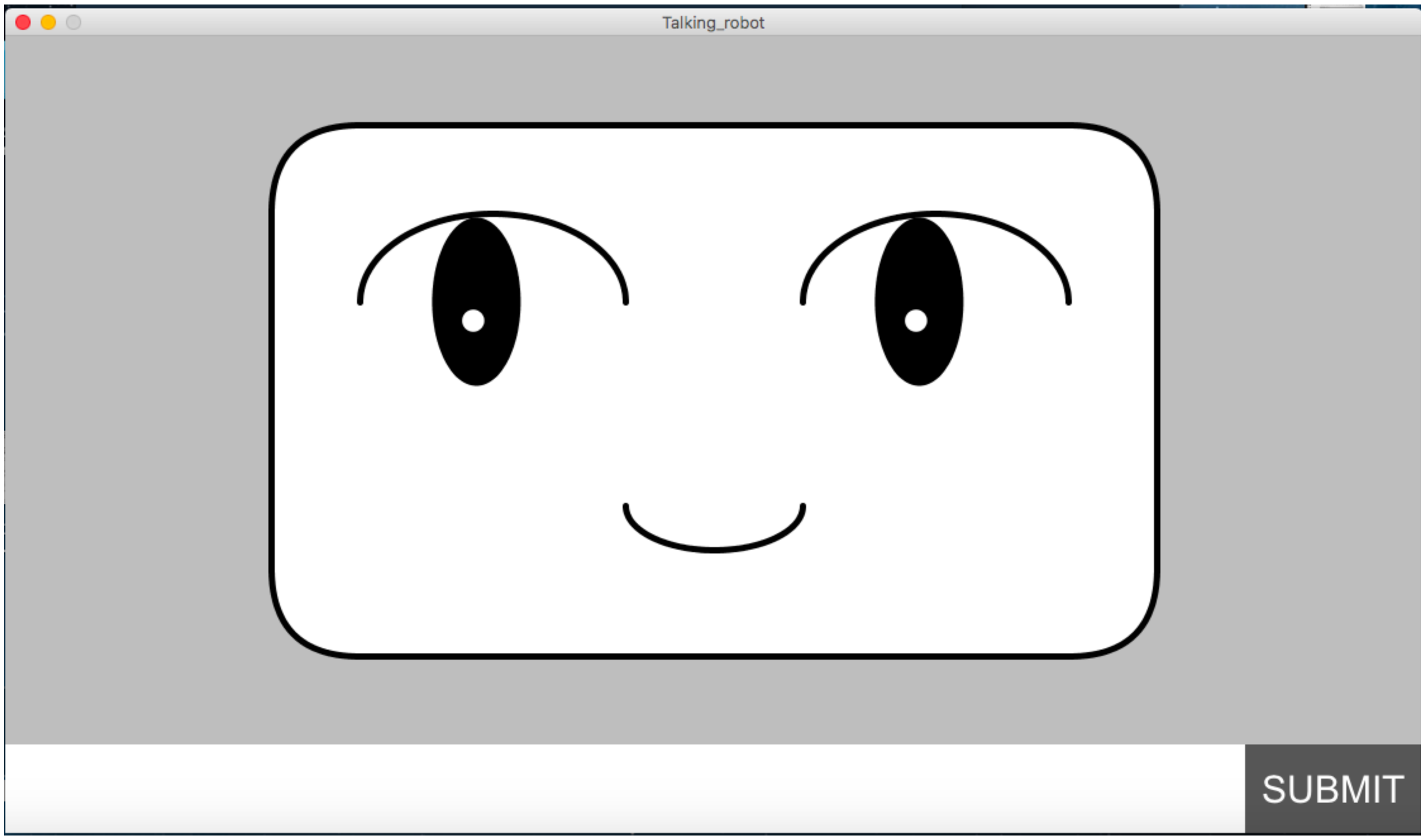
Additional Libraries

The built-in functions that we've called in the codes were written and defined by someone else earlier. The code for the functions are stored in the files associated with the Processing 3.0 software. However, there are tons of other functions and features that are not included in the software. For example, if you want to make a text box and make the robot talk, you can't really call such functions yet because they don't exist in the default Processing software on your computer. Fortunately, someone has already built these features and stored them in separate Processing libraries - you just have to download and import them into your own code. Let's say we want to build a textbox under the robot face, where users can type whatever they want. Once submitted, the robot will open its mouth and say whatever is in the textbox.

ControlP5

ControlP5, a library written by Andreas Schlegel, allows us to build user interface such as textbox and buttons in Processing. If you go to `Sketch->Import Library->Add Library`, and search for `ControlP5`, you can intall it on your computer and use it in your code. To build a textbox and a submit botton, the code will look like this:

```
import controlP5.*;
ControlP5 cp5;
float X;
float Y;

void setup(){
  size(1120,630);
  X=width/16;
  Y=height/9;
  cp5=new ControlP5(this);
  PFont font = createFont("arial", 30);

  cp5.addTextfield("")
     .setPosition(0,8*Y)
     .setSize(980,70)
     .setFont(font)
     .setColor(0)
     .setColorBackground(0xffFFFFFF)
     .setColorForeground(0xffFFFFFF)
     .setColorActive(0xff000000)
     .setAutoClear(false);

  cp5.addButton("SUBMIT")
     .setPosition(14*X,8*Y)
     .setSize(140,70)
     .setFont(font)
     .setColorLabel(0xffFFFFFF)
     .setColorBackground(0xff565656)
     .setColorForeground(0xff7C7C7C)
     .setColorActive(0xff565656)
     .getCaptionLabel()
     .align(ControlP5.CENTER, ControlP5.CENTER);
}
```



TTSLib

TTSLib is another library built by Nikolaus Gradwohl to help make Processing sketches speak. To import the library, the code will look like this:

```
import controlP5.*;
import guru.ttslib.*;
ControlP5 cp5;
TTS tts;
float X;
float Y;

void setup(){
  size(1120,630);
  X=width/16;
  Y=height/9;
  tts = new TTS();
  cp5=new ControlP5(this);
  PFont font = createFont("arial", 30);

  cp5.addTextfield("")
     .setPosition(0,8*Y)
     .setSize(980,70)
     .setFont(font)
     .setColor(0)
     .setColorBackground(0xffFFFFFF)
     .setColorForeground(0xffFFFFFF)
     .setColorActive(0xff000000)
     .setAutoClear(false);

  cp5.addButton("SUBMIT")
     .setPosition(14*X,8*Y)
     .setSize(140,70)
     .setFont(font)
     .setColorLabel(0xffFFFFFF)
     .setColorBackground(0xff565656)
     .setColorForeground(0xff7C7C7C)
     .setColorActive(0xff565656)
```

```
      .getCaptionLabel()
      .align(ControlP5.CENTER, ControlP5.CENTER);
}
```

A Talking Robot Face

Now we have set up both libraries, but how can we connect them to allow the robot to say whatever is the in the textbox? Also, it would look more natural if the mouth moves while the robot speaks. In this case, we need to use **boolean** variables. As discussed in the previous section, a boolean variable can only have to values: `true` or `false`. So the program can take a path. Under the `true` condition, some codes can be executed such as drawing an open mouth. However, we don't want to draw a open mouth while the robot is not talking, which will be part of the `false` condition:

```
import controlP5.*;
import guru.ttslib.*;
ControlP5 cp5;
TTS tts;
float X;
float Y;
boolean mouthOpen = false;
boolean speaking = false;

void setup(){
  size(1120,630);
  X=width/16;
  Y=height/9;
  tts = new TTS();
  cp5=new ControlP5(this);
  PFont font = createFont("arial", 30);

  cp5.addTextfield("")
      .setPosition(0,8*Y)
      .setSize(980,70)
      .setFont(font)
      .setColor(0)
      .setColorBackground(0xffFFFFFF)
      .setColorForeground(0xffFFFFFF)
      .setColorActive(0xff000000)
      .setAutoClear(false);

  cp5.addButton("SUBMIT")
      .setPosition(14*X,8*Y)
      .setSize(140,70)
      .setFont(font)
      .setColorLabel(0xffFFFFFF)
      .setColorBackground(0xff565656)
      .setColorForeground(0xff7C7C7C)
      .setColorActive(0xff565656)
      .getCaptionLabel()
      .align(ControlP5.CENTER, ControlP5.CENTER);
}

void draw(){
  background(190);
  faceOutline();
  eyesBlink();
  mouth();
  thread("ttsSpeak");
}

void faceOutline(){
  stroke(1);
  strokeWeight(5);
  fill(255);
  rect(3*X, Y, 10*X, 6*Y, X);
}
```

```
void eyesBlink(){
  if(frameCount % 350>0 && frameCount % 350 <25) {         //eyes open every 350 frames and stay open for 25 frames
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, 0,PI);                      //right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, 0,PI);                     //left eyelid

  }else if(mouseX<14*X || mouseY<8*Y){                     //eyes open with moving eyeballs if the mouse position is outside the S
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                  //right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                 //left eyelid

    fill(0);
    noStroke();
    float eyeballA=map(mouseX,0,width,5*X,6*X);
    ellipse(eyeballA,3*Y,X,1.9*Y);                         //right big eyeball
    ellipse(eyeballA+5*X,3*Y,X,1.9*Y);                     //left big eyeball

    fill(255);
    noStroke();
    float eyeballBX =map(mouseX,0,width,4.9*X,6.1*X);
    float eyeballBY =map(mouseY,0,height,2.25*Y,3.75*Y);
    ellipse(eyeballBX,eyeballBY,X/4,Y/4);                  //right small eyeball
    ellipse(eyeballBX+5*X,eyeballBY,X/4,Y/4);              //left small eyeball

  }else{                                                    //eyes open with static eyeballs if the mouse position is inside the SU
    strokeWeight(5);
    arc(5.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                  //right eyelid
    arc(10.5*X, 3*Y, 3*X, 2*Y, PI, 2*PI);                 //left eyelid

    fill(0);
    noStroke();
    ellipse(5.5*X,3*Y,X,1.9*Y);                            //right big eyeball
    ellipse(10.5*X,3*Y,X,1.9*Y);                           //left big eyeball

    fill(255);
    noStroke();
    ellipse(5.5*X,3*Y,X/4,Y/4);                            //right small eyeball
    ellipse(10.5*X,3*Y,X/4,Y/4);                           //left small eyeball
  }
}

void SUBMIT(){                                              //when the SUBMIT button is released, speaking and mouthOpen return tru
  mouthOpen = true;
  speaking = true;
}

void mouth(){
  if (mouthOpen && frameCount % 40 >0                      //if mouthOpen is true, mouth opens every 40 frames and stays open for
      && frameCount % 40 <15){
    fill(255);
    stroke(1);
    strokeWeight(5);
    arc(8*X,5.3*Y,2*X,2*Y,0 ,PI);                          //lower lip

    stroke(1);
    fill(252,50,111);
    arc(8*X,5.3*Y,2*X,2*Y,0.25*PI ,0.75*PI,OPEN);          //tongue

    fill(255);
    arc(8*X,5.3*Y,2*X,0.5*Y,PI,2*PI);                      //upper lip
  }else{                                                    // if mouthOpen is false, mouth closes
    fill(255);
    stroke(1);
    strokeWeight(5);
    arc(8*X,5.3*Y,2*X,Y,0 ,PI);
  }
}

void ttsSpeak(){                                          //if speaking is true, speaking returns false, the robot speaks, and th
  if (speaking){
    speaking = false;
    tts.speak(cp5.get(Textfield.class,"").getText());
    mouthOpen = false;
  }
}
```

And there you have it - a talking robot face built in Processing 3.0. Now go forth, take what you have learned in this tutorial and make some amazing scripts of your own!