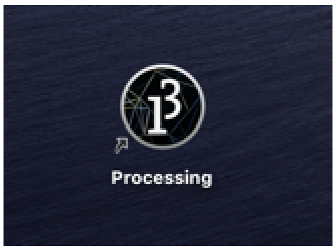
Lesson **1**

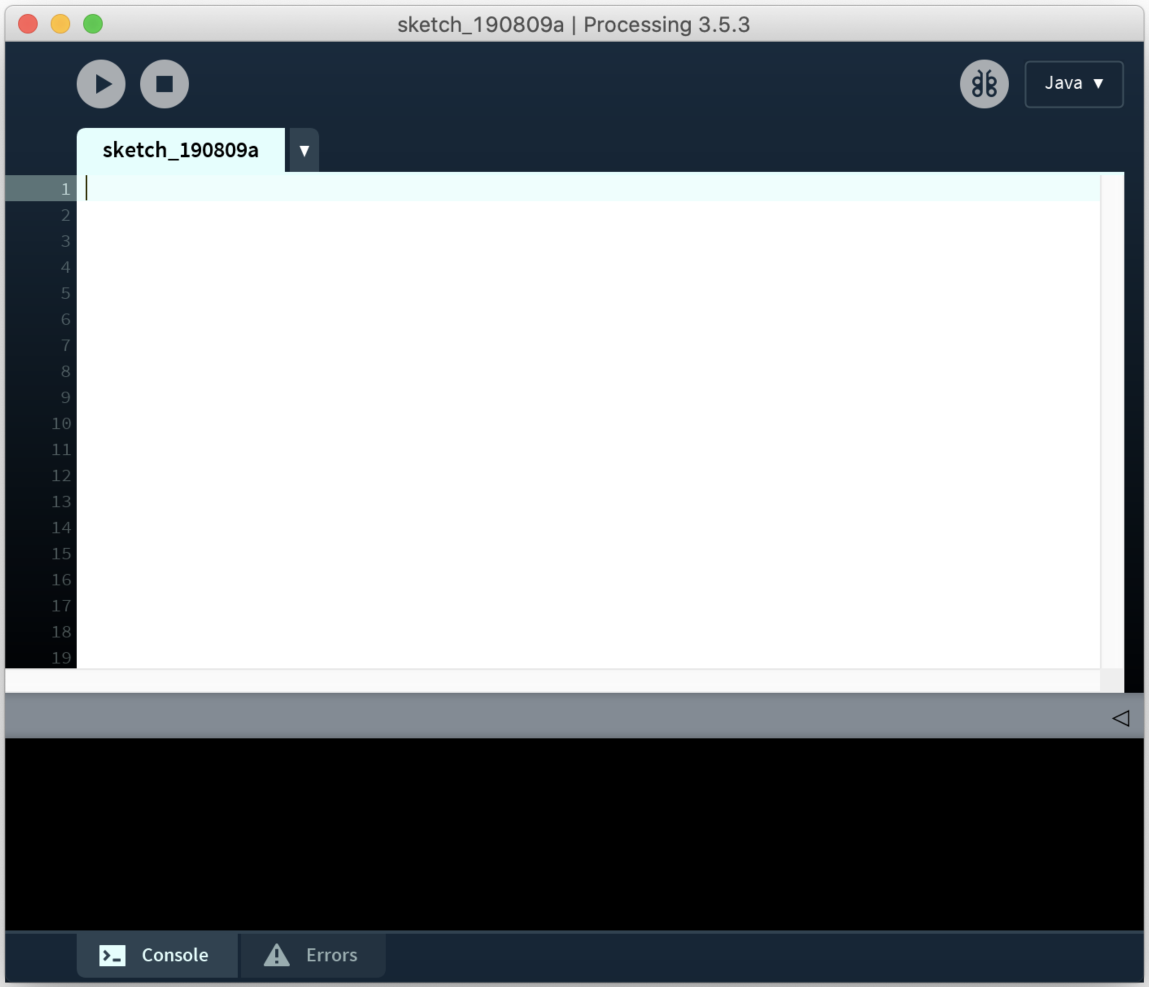
Opening Processing

We will be using a program called "Processing" to write our programs. Before we can write anything, we need to open this program.

Look for the "Processing" icon on the desktop. It should look like this:



Double click on the icon to open Processing. Once it has loaded, you should see the "Processing" text editor on your screen:



We use the Processing program to write our own programs. To write a program, we click on the white area and type our program. Once we have finished writing our program, we click on the ▶ Play button to run it.

Screenshot of processing features

Tip:
If you are typing and nothing shows up, make sure it is *focused* by clicking your cursor anywhere on the white part (this is the text editor) to select it. You should now be able to type into it.

Congratulations! We are now ready to start making our first Processing program!

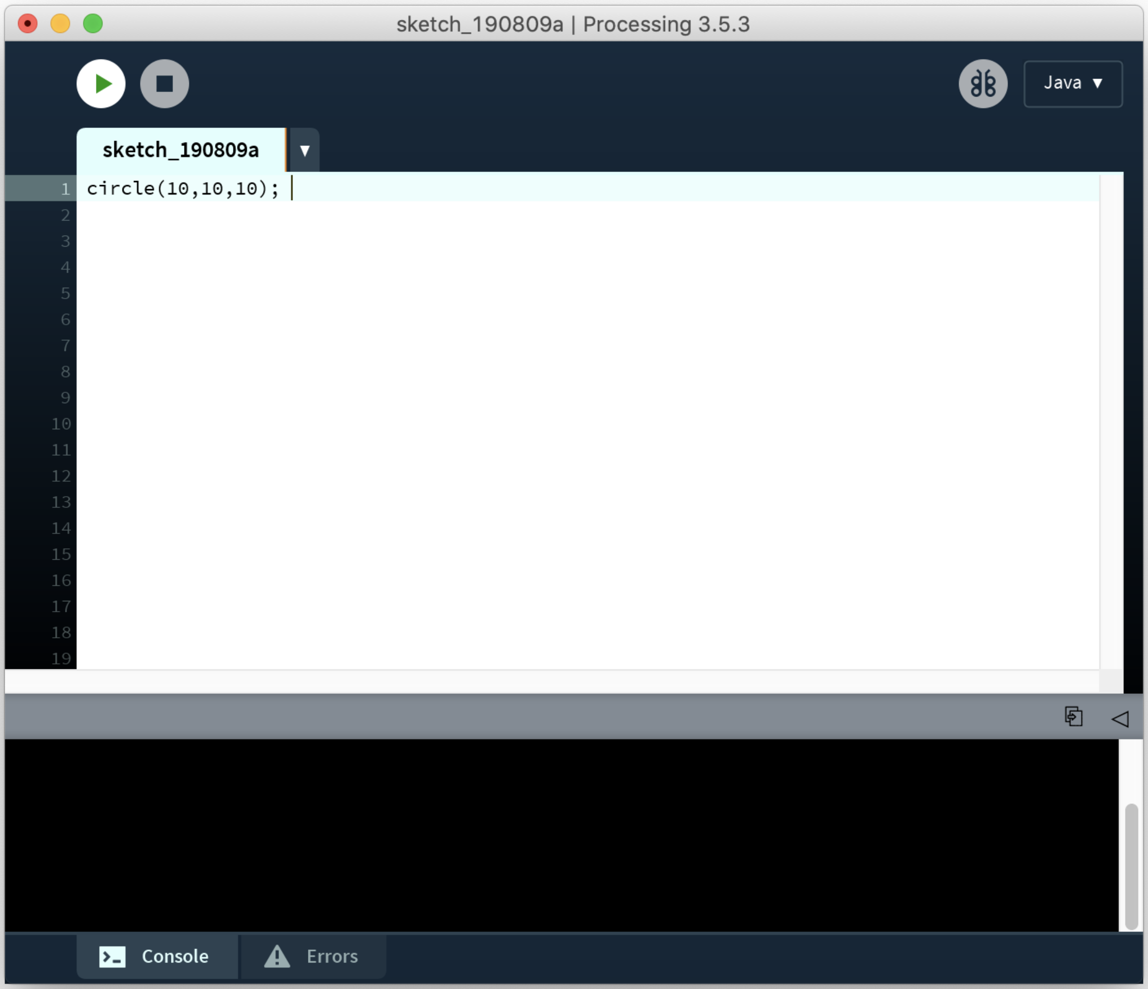
Lesson **2**

Our first program

Let's write our first program! Type the line below into the Processing text editor:

```
ellipse(10,10,10,10);
```

You should see this on your screen:

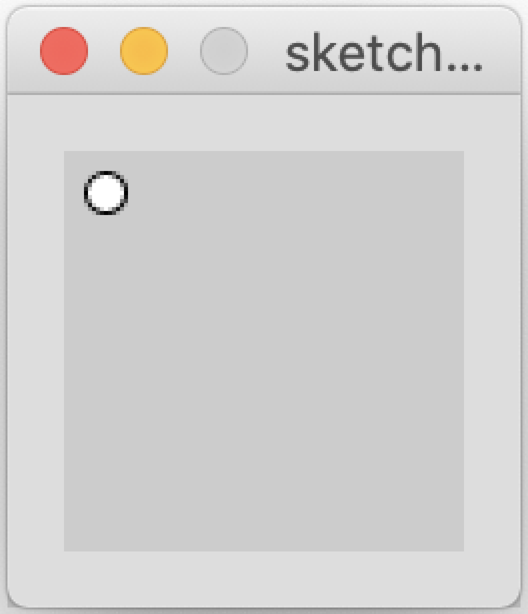

sketch_190809a | Processing 3.5.3

```
circle(10,10,10);
```

> 💡 **Tip:**
> If you are typing and nothing shows up, make sure it is *focused* by clicking your cursor anywhere on the white part (this is the text editor) to select it. You should now be able to type into it.

Let's run your program! Click on the ▶ Play icon on the top of the Processing window:

You should see a new window with a circle in it:



🎓 *Congratulations!* You just wrote your first program.

It may not make any sense to us right now, but let's try to figure out what we just did.

The computer does not know what we want to do unless we give it instructions. The line we just typed is an instruction that tells it to draw an `ellipse` (ellipses are round shapes like circles or ovals).

Great! Now the computer knows we want to draw an `ellipse`, but we still need to give it more instructions! If we didn't, it wouldn't know how big to make our circle or where to put it. Computers don't like guessing!

Illustration of several different circles

That's where the second part of our instruction comes in - the numbers that look like `(10,10,10,10)`. In programming, we call these *arguments*, but they are just instructions that tell the program where to put the circle and how big to make it.

Finally, we need to tell the program that we are finished giving it an instruction. You might use a period when you are writing a sentence, but computers use a different letter - the semicolon: `;`. It's a weird letter, but that's what we use to tell programs we are finished with our instructions.

So, what did we do when we wrote this line? We gave the program a *function* (draw an `ellipse`), a bunch of *arguments* `(10,10,10,10)`, and then told it we are finished with our instruction `;`

Components of a function

We don't know what the numbers do yet, but let's try changing them! Change the numbers to something different (but keep each number under 100 for now), then click the Play button. What happens to your circle?

Exercise:
Try changing the numbers to something else like (30,25,32,48), then click Play. What happens to your cicle? Try changing the number a few times!

Lesson **3**

Drawing more shapes

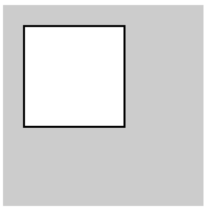
We just drew a circle, but there are so many other types of shapes we can draw!

Rectangles and Squares

Let’s make a square! Press the ‘Backspace’ key until all of your code is removed, and then thype the line below:

```
rect(10,10,50,50);
```

Now press the ▶ **Play** button. You should see this on your screen:



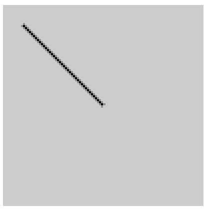
This program looks a lot like our first program, but we replaced the `ellipse` *function* with the `rect` *function*. We have now told the computer we want to draw a rectangle instead of a circle.

Lines

Let’s try a different function! Replace the `rect` function with `line`. Your program should look like this:

```
line(10,10,50,50);
```

Now press the ▶ **Play** button. You should see this on your screen:



Triangles

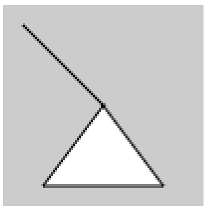
Let’s try drawing a triangle! Press the ‘Enter’ key, then type the line below:

```
triangle(50,50,20,90,80,90);
```

Your program should look like this:

```
line(10,10,50,50);
triangle(50,50,20,90,80,90);
```

Now press the ▶ **Play** button. You should see this on your screen:



We just did a couple of interesting new things! All of the *functions* we have written until now have four numbers, but the *triangle function* has six! Some functions - like *triangle* - need more information than others.

The other thing we just did is we wrote a program that has two instructions. When we run the program, the computer will first create a *line* and then create a *triangle*. This is why we need to type the `;` semicolon at the end of each instruction, so the computer knows when to start the next instruction. You would think it could figure this out every time you type a new line, but computers aren't very smart.

Learning more about our functions

We're starting to see a bunch of *function*s! We've tried `ellipse`, `rect`, `line`, and `triangle`. There are many more *function*s in programming, but they can be a little hard to remember. From now on, I will add a little guide every time we discover a new function. If you click on it, you can see more information and some examples:

New Function: ellipse This function lets you draw an 'ellipse'. Ellipses can be circles, but they can also be squished circular shapes like ovals

New Function: rect This function lets you draw an rectangle. A square is also a rectangle, but it has the same height and width.

New Function: line This function lets you draw a line

New Function: tringle This function lets you draw a triangle

We've tried a bunch of functions, but each one of them has these weird numbers like `(10,10,10,10)` next to it. What do those mean? Let's look at that next.

Lesson **4**

Placing things on the screen

So far, we've written a number of *functions* - like `ellipse` - with a bunch of numeric *arguments* - such as `(10,10,10,10)` - that tell the program where to put the shapes and how big to make them. But what do these numbers mean?

The screen you see in front of you is made up of millions of teeny, tiny lights. When we write a program, we tell the computer what to do with these lights. But in order to do this, we need a way to tell the computer *which* lights we want to control.

If you hold a magnifying glass in front of a screen, you will see that it looks a little bit like this:

Illustration of graph paper

We need to pick a number for each light. The easiest way to do this (and the way programs do it) is by starting from the left side of the screen and giving the first column the number 1. We give the second column the number 2. The third is 3, and so on. So it looks a bit like this:

Illustration of graph paper with columns numbered

But this isn't quite good enough! We can pick the column we want to control, such as column 3:

Illustration of graph paper with columns numbered and 3 highlighted

But what if we want to control a specific light? To do this, we also number each row starting from the top:

Illustration of graph paper with columns and rows numbered

So each light actually has two numbers - the column and the row. So, the number in black below...

Illustration of graph paper with columns numbered and 3,2 highlighted

..is in the *3rd colummn from the left*, and the *2nd column from the top*.

To do this, we give each light a number. Actually, we give it two - a number from the left, and a number from the top. To figure out the number for each light, we start from the top left side of the screen. The light at the top left

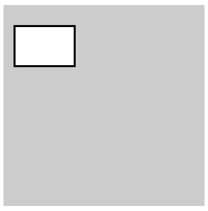
We can also use the same numbers to tell the computer how wide or tall to make something. So, for example, you could say "**draw a rectangle that starts from the 5th light on the left and the 10th light on the top, and make it 30 lights wide and 20 lights tall**".

This is exactly what we do when we write the following instruction:

Code:

```
rect(5,10,30,20);
```

Preview:



The four *arguments* we give the `rect` *function* are instructions on where to start the rectangle on the left and top, and how wide and tall to make it:

Breakdown of arguments

Exercise:
What would the four (left, top, width, height) arguments for this rectangle be? *Image of rectangle on graph paper*

Exercise:
Draw your own rectangle on the graph paper below, then write the *rect* function that would create it. *Image of blank graph paper*

The *line* function is a little like from the *rect* function. The first two arguments tell us where to start the line from the left of the screen and the top of the screen. The third argument tells us how wide to make the line, and the fourth argument tells us how tall to make the line:

Image of Line with points

The *ellipse* function is a little different, because it doesn't have a top and left point - it's a circle! Instead, we tell the program where to put the middle of the circle from the left and top of the screen, as well as how wide and tall to make it:

Image of ellipse with points

The *triangle* function is the weirdest! Triangles have three points, and so we need to tell the program where to put each of them. The first two arguments tell the program where to put the first point from the left and top of the screen, the second two arguments tell us where to put the second point from the left and top of the screen, and the last two arguments tell us where to put the third point from the left and top of the screen:

Image of triangle with points

Tip:
Don’t worry if you are having a hard time figuring out which numbers to write down! A lot of programmers don’t know what they are doing when they type in the numbers. You can always try random numbers until you find something that looks like what you want. The more you play around, the more your brain will be able to guess what the numbers should be..

Exercise:
Create a program that draws a face on the screen. You will need to draw two circles - one for each eye - and a rectangle for the mouth. If you want, you can also draw a triangle for the nose!

Lesson **5**

Black, White, and In Between

So far, all of our shapes have looked pretty boring. Let's learn how we can add some color to them!

Shapes have a solid center and an edge, and both of these can be colored. We use two *functions* to do this: `fill` to change the middle, and `stroke` to change the edge.

Circle with arrows pointing to center and edges referencing fill and stroke

When we talk about colors, we use words like 'red' and 'pink', but computers don't know what these words mean. Instead, we have to use numbers to tell the computer what color to use. We pass these numbers as *attributes* to the `fill` and `stroke` functions.

Let's take a look at how we do this. Type the following line at the top of your program:

```
fill(0);
```

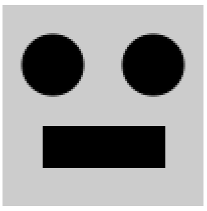
We are telling the program to run the *function* `fill` with an *attribute* of `0`.

If you just drew a face at the end of the previous lesson, your code might look a little bit like this:

Code:

```
fill(0);
ellipse(25,30,30,30);
ellipse(75,30,30,30);
rect(20,60,60,20);
```

Preview:



When we typed `fill(0);` at the top of the program, we told the computer to `fill` all of the shapes after it with a color of `0`. The color `0` is black, so the eyes and mouth will now look black.

Tip:
We have to tell the program which color to pick *before* we draw any shapes, kind of like how you need to pick a color before you draw something on a piece of paper. If your face's color is still white, make sure that you put the *fill* function at the top of the program.

Colors are expressed as numbers from 0 to 255. 255 is a weird number, but it is a weird quirk with how computers store numbers. You will see that number quite a lot when you write programs!

The color 0 is black, and the color 255 is white. Every color between white and black has a number between 0 and 255. Here is what some of these numbers look like:

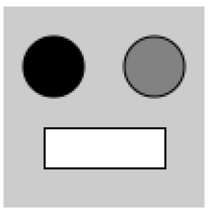
Color	Number	Code
⬛ Black	0	`fill(0);`
▨ Dim Gray	105	`fill(105);`
▨ Gray	128	`fill(128);`
▨ Dark Gray	169	`fill(169);`
▨ Silver	192	`fill(192);`
▨ Light Gray	211	`fill(211);`
▨ Gainsboro	220	`fill(220);`
⬜ White Smoke	245	`fill(245);`
White	255	`fill(255);`

Let’s try changing the program so that we have different colors for each part of the face:

Code:

```
fill(0);
ellipse(25,30,30,30);
fill(130);
ellipse(75,30,30,30);
fill(255);
rect(20,60,60,20);
```

Preview:



Lesson **6**

Color

There's one really big problem here. We haven't really made anything colorful yet! How do we make things actually look colorful?!

So far, we've been using the `fill` function with one attribute:

```
fill(grayscale);
```

But `fill` can also take three attributes, with numbers between 0-255 for red, green, and blue:

```
fill(red, green, blue);
```

The way we make colors is by mixing red, green, and blue values.

If we wanted to make red, we would give it a `(red, green, blue)` value of `(255,0,0)`. This tells the program to make the color as red as possible (255), and no green (0) and no blue (0).

To make green, we would give it a `(red, green, blue)` value of `(0,255,0)`: no red, as much green as possible, and no blue.

To make bleu, we would give it a `(red, green, blue)` value of `(0,0,255)`: no red, no green, and as much blue as possible.

So to make red, green, and blue, we would type these lines:

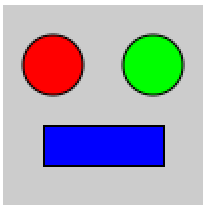
Color	**Red**	**Green**	**Blue**	**Code**
🟥 Red	255	0	0	`fill(255,0,0);`
🟩 Green	0	255	0	`fill(0,255,0);`
🟦 Blue	0	0	255	`fill(0,0,255);`

Here's what this might look like in a program

Code:

```
fill(255,0,0);
ellipse(25,30,30,30);
fill(0,255,0);
ellipse(75,30,30,30);
fill(0,0,255);
rect(20,60,60,20);
```

Preview:



When you are painting, you can mix red, green, and blue together to make different colors. The same thing happens in programs too. Here are some common color mixtures:

Color	Red	Green	Blue	Code
🟨 Yellow	255	255	0	`fill(255,255,0);`
🟦 Cyan	0	255	255	`fill(0,255,255);`
🟪 Fuchsia	255	0	255	`fill(255,0,255);`
🟪 Purple	128	0	128	`fill(128,0,128);`
🟦 Navy	0	0	128	`fill(0,0,128);`
🟥 Crimson	220	20	60	`fill(220,20,60);`
🟨 Gold	255	215	0	`fill(255,215,0);`
🟥 Deep Pink	255	20	147	`fill(255,20,147);`
🟫 Wheat	245	222	179	`fill(245,222,179);`
🟦 Steel Blue	119	196	222	`fill(119,196,222);`
🟩 Sea Green	46	139	87	`fill(46,139,87);`

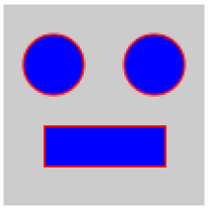
Exercise:
Change the color of your face so that it is your favorite color. You can use one of the colors from the table above, or try typing in different numbers until you find a brand new color you like!

We can also change the color of the line around each shape with the `stroke` *function*. This function uses the same attributes as `fill`, so if we wanted to make a blue face with red lines we might type something like this:

Code:

```
fill(0,0,255);
stroke(255,0,0);
ellipse(25,30,30,30);
ellipse(75,30,30,30);
rect(20,60,60,20);
```

Preview:



Tip:
Don't worry if its hard to guess which *attributes* to type to get the color you want. Most programmers don't know what numbers to type, and use a reference guide to tell them what to type in. You can also experiment by typing different numbers for each attribute until you get a color you like!

Lesson **7**

Variables

Until now, we have always typed the numbers for our *attributes* directly, like this:

```
fill(255);
```

But in most programs, we don't do this. Instead, we usually use something called a *variable*. When you write a *variable*, you can think of it as the computer remembering a number for you. When you want, you can ask the program the write the number back. Here's how the same program looks with a *variable*:

```
int color = 255;
fill(color);
```

When we create a variable, we pick a name. We can give a variable any name we want, but it should only have letters - no numbers or spaces. Names are normally uncapitalized (e.g. `age` instead of `Age`), which might make your English teacher upset but makes programmers happy.

Not being able to use spaces is annoying, but there are a few ways we can deal with this. If we want to use a name with multiple words, such `My Favorite Color`, we either use Capital Letters to indicate the words: `myShoeSize`, or we use the underscore (`_`) letter: `my_shoe_size`. Either option works, so it comes down to which one you find prettier.

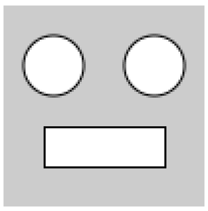
Here's what some variables might look like:

```
int color = 255;
int myFavoriteColor = 255;
int age = 15;
int shoe_size = 8;
```

Code:

```
ellipse(25,30,30,30);
ellipse(75,30,30,30);
rect(20,60,60,20);
```

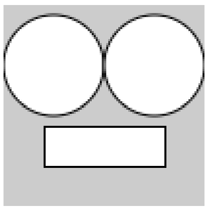
Preview:



Code:

```
ellipse(25,30,50,50);
ellipse(75,30,50,50);
rect(20,60,60,20);
```

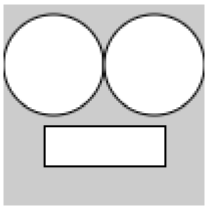
Preview:



Code:

```
int eyeSize = 50;
ellipse(25,30,eyeSize,eyeSize);
ellipse(75,30,eyeSize,eyeSize);
rect(20,60,60,20);
```

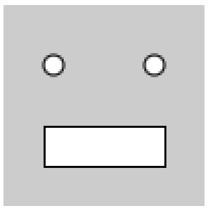
Preview:



Code:

```
int eyeSize = 10;
ellipse(25,30,eyeSize,eyeSize);
ellipse(75,30,eyeSize,eyeSize);
rect(20,60,60,20);
```

Preview:



Code:

```
int eyeSize = 20;
int left = 20;
ellipse(left + 5,30,eyeSize,eyeSize);
ellipse(left + 50,30,eyeSize,eyeSize);
rect(left,60,60,20);
```

Preview: