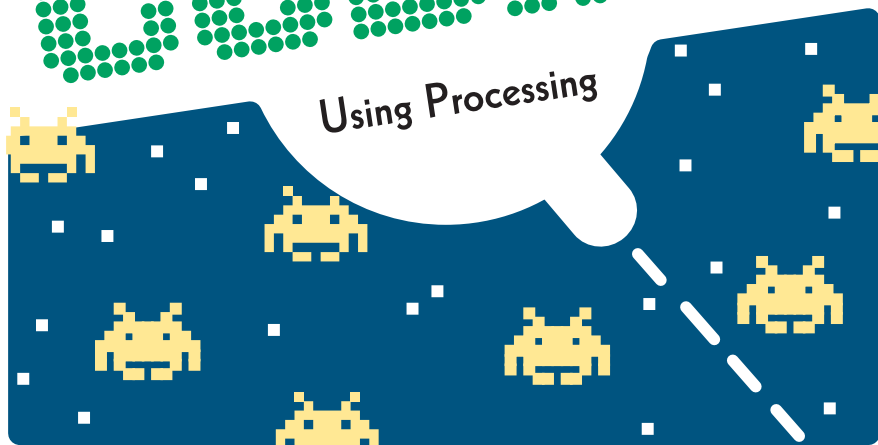


The Absolute Beginner's Guide To

CODING

Using Processing



Index

01	Why Learn To Code?	4 - 5
	All About Programming • Downloading Processing	
02	Create Your First Program	6 - 8
	Learn about the Processing Development Environment	
03	Commenting Your Code	10
	Single Line Comments • Multiple Line Comments	
04	Making Shapes	11-15
	Ellipses • Rectangles • Triangles • Processing Reference Page	
05	Colors	16-21
	RGB Colors • Fill Color • Background Color • Stroke Color	
06	Variables	22-27
	Declaring Variables • Swapping Values	
07	Printing Values	28-31
	print() • println() • Newline • Concatenation	
08	Doing Math with Variables	32-35
	Addition • Subtraction • Multiplication • Division	

09	If / Else Statements	36 - 39
	Operators • If Statement • Else Statement • Else If Statement	
10	While Loops	40 - 44
	Repeating Code	
11	For Loops	10
	Repeating Code • For Loops • Nested For Loops	
12	Arrays and ArrayLists	11-13
	Integers • Floats • Booleans • Color	
13	Functions	11-13
	Creating Classes and Objects	
14	Classes and Objects	11-13
	Key pressed function	
15	MouseX and MouseY	11-13
	mouseX and mouseY	
16	Other	11-13
	Declaring ArrayL	

Why Learn To Code?

01

All About Programming

Have you ever wondered how websites are built, how video games are created, or how apps for smartphones are designed?

They were all made with a programming language, which is a special language that allows people to communicate with computers.

What is Processing?

One of the many programming languages available is Processing. People like to program with Processing because it is easy to draw images, so it is great for making video games and animations.

When you write a program in Processing, you are making a list of instructions for the computer to follow. A computer's job is to follow instructions, in fact a computer follows millions of instructions every second.

Your job is to learn how to communicate with the computer so it will follow your instructions. Don't worry, learning how to code is not hard at all, and you will soon be well on your way to creating your own computer programs.

Download Processing

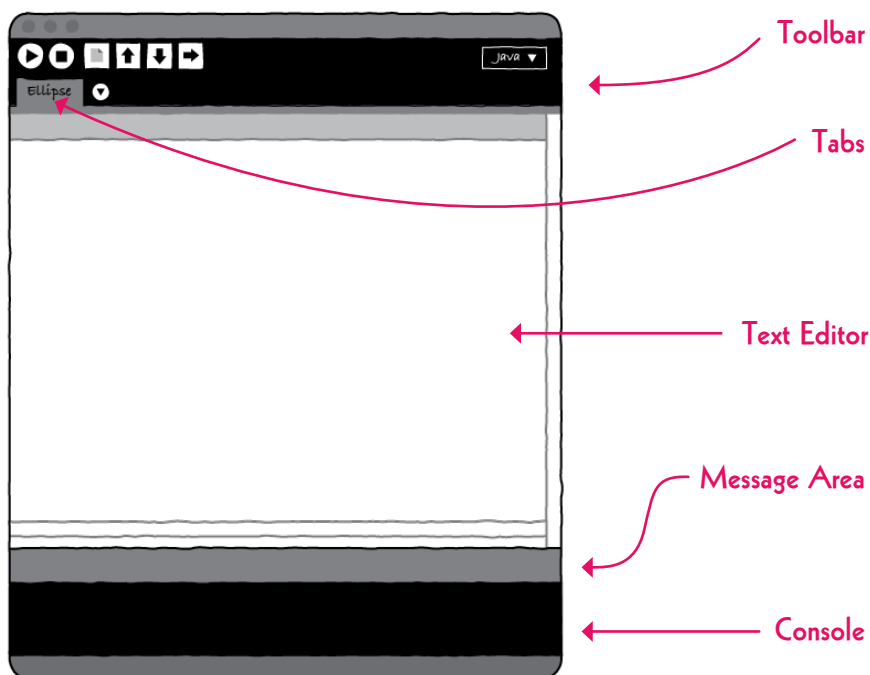
Processing is a free program that can be downloaded for your home computer from www.processing.org.

Processing is available for Macs, Windows, and Linux.

Create Your First Program

02

When you first open Processing you will see something similar to the sketch below.



Step 1) The first thing you should do every time you create a new project is to save (**File > Save**) it to some location on your computer.

Step 2) Lets draw a circle. In the text editor of Processing type the following:


```
ellipse(50, 50, 80, 80);
```

Step 3) When we type “ellipse(a, b, c, d),” we are calling a function. A **function** is code that has already been written for us by the people who created Processing, so if we want to draw a circle all we have to do is call the ellipse() function with the proper **parameters**. An ellipse function has 4 parameters. Other functions may have a different number of parameters.

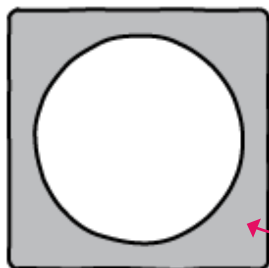
Function Name

ellipse(x-coordinate, y-coordinate, width, height);

Parameters

 **Step 4)** To run the program, click the Run button, which is a triangle that looks like a Play button located on the far left of the Toolbar. Another window called the Display window will launch.

Hint: If your program does not run it is probably because you did not copy the ellipse function exactly as you see it above. It is important to copy everything just as you see it, even parenthesis and semicolons. If you do not, the program may complain or it may simply not run.



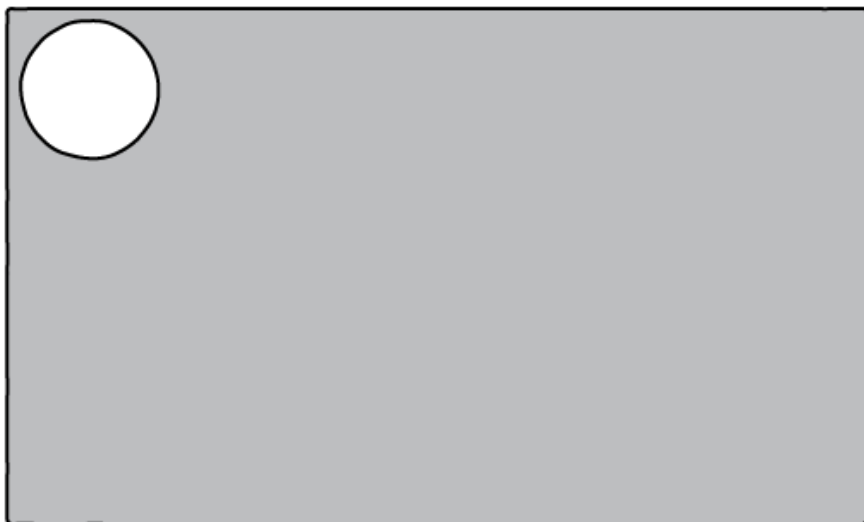
Step 5) The Display window is kind of tiny. Lets make it bigger. Lets go back to the text editor and add the following:

The display window will pop up and show an ellipse

```
size(500, 300);  
ellipse(50, 50, 80, 80);
```

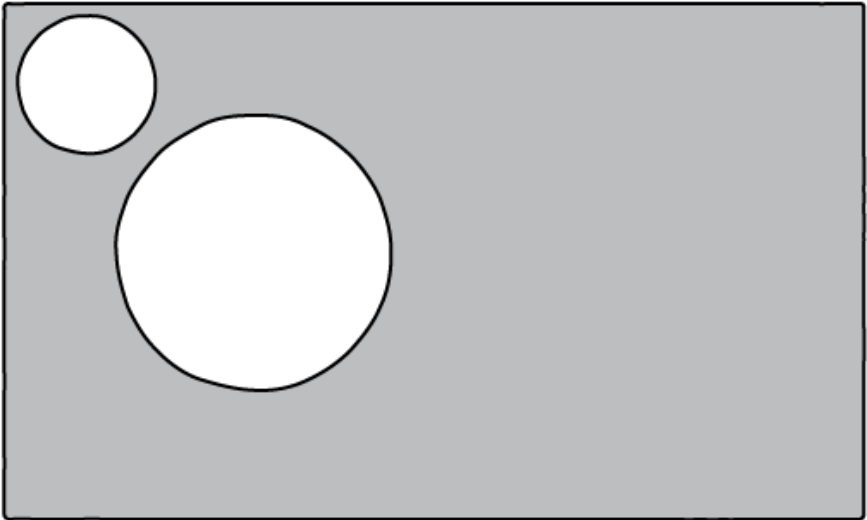
Step 6) You may have noticed that both lines in the above program end with a semicolon(;). In Processing, every statement must end with a semicolon. When you first start programming, it is very easy to forget the semicolon, but as you gain more practice, adding the semicolon to the end of the line will become second nature.

Step 7) Click the Run button again. The size of the display window is now 500 pixels wide and 300 pixels high.



Step 8) Lets draw a second ellipse.

```
size(500, 300);  
ellipse(50, 50, 80, 80);  
ellipse(150, 150, 160, 160);
```



Step 9) Whew, we just finished our first computer program. In the next chapters we will learn how to create many different shapes.

Commenting Your Code

03

What Are Comments? One of the most important parts of programming is commenting on the code you created. Comments are not part of the code, in fact the computer completely disregards any comments. Comments are for your own use, reminding you of your thought process as you created the code.

Why Comment? It may seem obvious why you coded something the way you did now, but if you reopen the program a year or two from now, you probably won't remember why you coded something in a certain way.

Comment One Line To comment out one line of text, use double slashes (//) in front of your comment.

Comment Multiple Lines To comment out multiple lines, put your comments between /* and */.

```
// Draw a circle  
ellipse(50, 50, 80, 80);  
/* Draw a second circle  
   The second circle is twice as big as the first */  
ellipse(150, 150, 160, 160);
```

Making Shapes

04

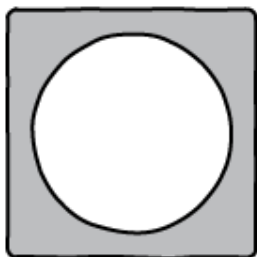
Circles and Ellipses

An **ellipse** looks like a squashed circle. A **circle** is a special kind of ellipse that has the same width and height.

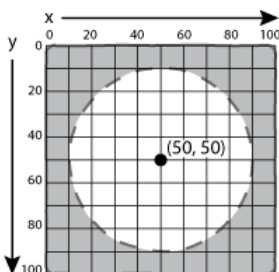
`ellipse(x-coordinate, y-coordinate, width, height);`

center of the ellipse

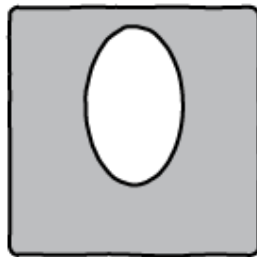
dimensions of the ellipse



`ellipse(50, 50, 80, 80);`



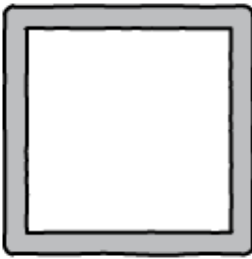
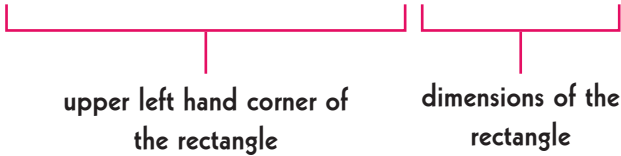
The coordinate system



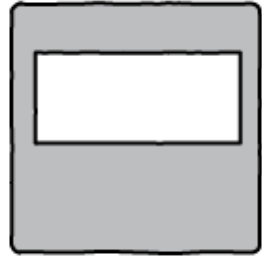
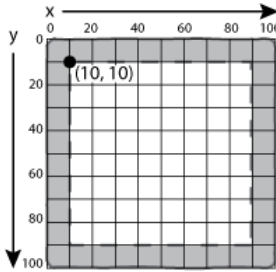
`ellipse(50, 40, 35, 70);`

Rectangles

`rect(x-coordinate, y-coordinate, width, height);`



`rect(10, 10, 80, 80);`



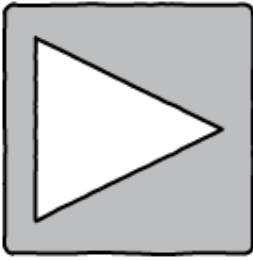
`rect(10, 20, 80, 40);`

Triangles

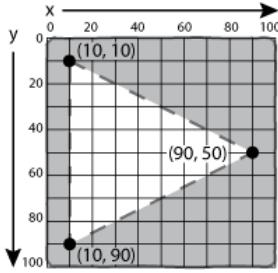
To create a triangle in Processing, you need to know the x and y-coordinates of each corner of the triangle.

`triangle(x-cord, y-cord, x-cord, y-cord, x-cord, y-cord);`

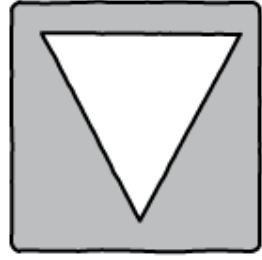




```
triangle(10, 10, 90, 50,  
10, 90);
```



The coordinate system



```
triangle(10, 10, 90, 10,  
50, 90);
```

Creating Other Shapes

You can create lots of other shapes in Processing, including arcs, lines, quads (otherwise known as shapes with 4 sides), and points. To learn how to create these shapes, look-up the shape's name on the Processing Reference page.

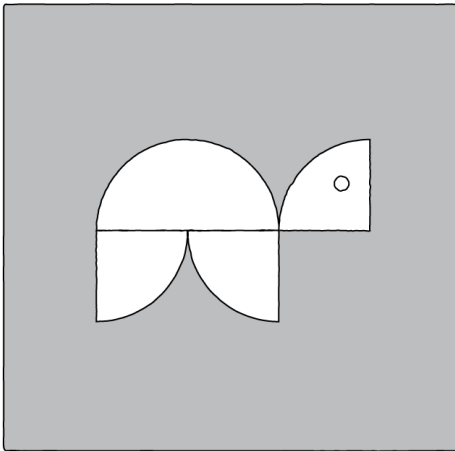
Processing Reference If you ever forget how to create a shape, the fastest way to get that information is from the Processing Reference website page. While you are in Processing, go to [Help > Reference](#). Your web browser will start and bring up the Processing Reference page. You will see a very long list words, which looks intimidating, but look for the headline Shape and then look for the subheading 2-D Primitives. Click on a function name (for example: “`ellipse()`”) to get more information about how to create the shape.

Example: Draw a Turtle

```
// Dimension of the window  
size(250, 250);
```

```
// The turtle's body is created out of 4 arcs  
arc(100, 125, 100, 100, PI, (PI * 2)); // Shell  
arc(50, 125, 100, 100, 0, (PI / 2)); // Left leg  
arc(150, 125, 100, 100, (PI / 2), PI); // Right leg  
arc(200, 125, 100, 100, PI, (PI / 2) * 3); // Head
```

```
// The turtle's eye is an ellipse  
ellipse(185, 100, 7, 7); // Eye
```



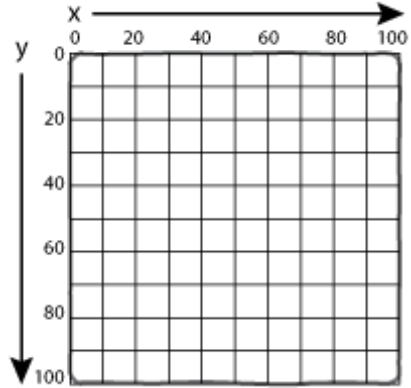
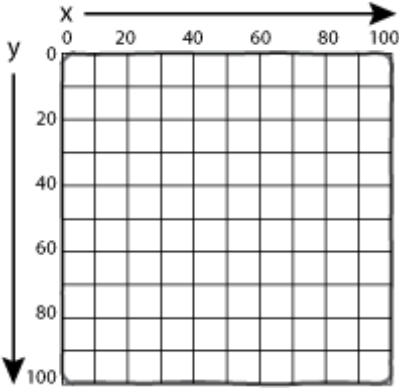
Hint: If you do not know how to make an arc shape, look up how to draw the shape on the Processing Reference Page.

Practice Problems

1) Can you draw these shapes?

```
ellipse(50, 50, 30, 80);
```

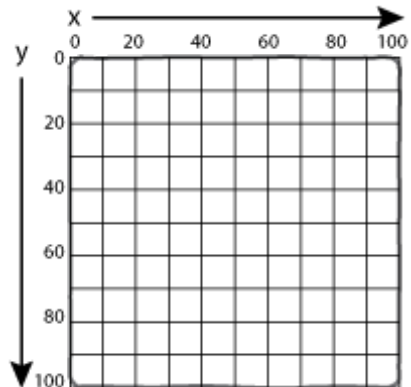
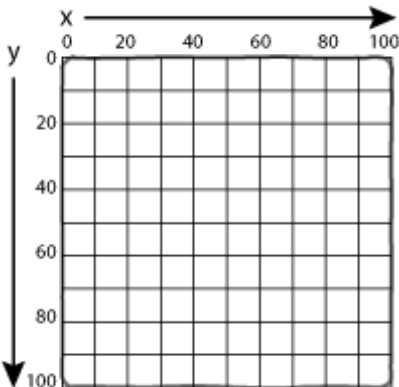
```
line(10, 10, 90, 90);
```



2) What do you think will happen if you draw a rectangle using `rectMode(CENTER)` or `rectMode(CORNER)`?

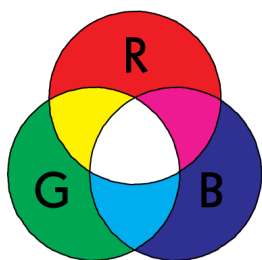
```
rectMode(CENTER);  
rect(50, 50, 80, 80);
```

```
rectMode(CORNER);  
rect(10, 10, 80, 80);
```



Creating Colors

05



Remember from art class, if you wanted to create orange paint you would mix yellow and red paints together. Creating colors on the computer is a little bit different, because when computers create colors they mix light, not pigments.

RGB Three colors of light: red, blue and green, can be mixed to create pretty much every color imaginable. The red, blue, and green recipe is known as a **RGB value**.

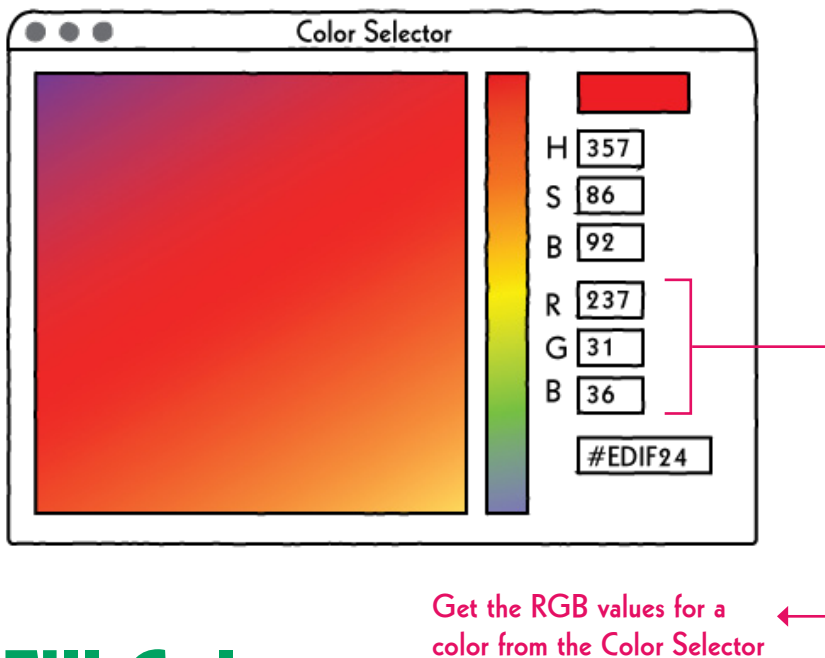
Color Functions Processing has three functions that allow you to color shapes easily, `fill()`, `stroke()`, and `background()`.

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

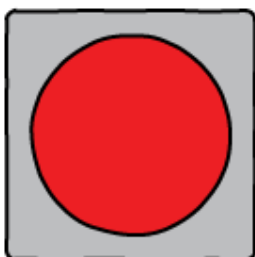


each parameter must be a number between 0 and 255

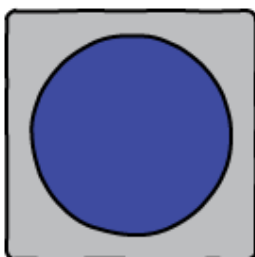
Finding RGB Values The easiest way to create a color is to find a color you like in a color palette and get the RGB value. In Processing, you can do this by opening the color selector (**Tools > Color Selector**). The RGB color value is broken into three numbers, each of which is between the values 0 and 255. Find a color that you like and copy the RGB values of the color into a color function.



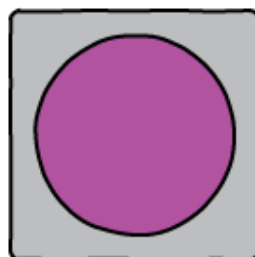
Fill Color



```
fill(237, 31, 36);  
ellipse(50, 50, 80, 80);
```

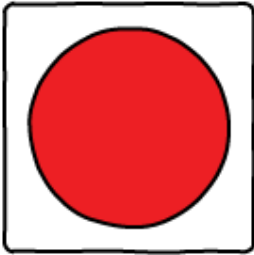


```
fill(63, 75, 160);  
ellipse(50, 50, 80, 80);
```

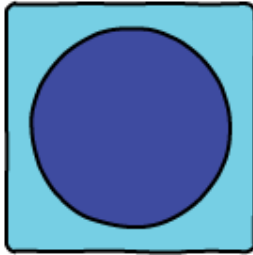


```
fill(117, 83, 160);  
ellipse(50, 50, 80, 80);
```

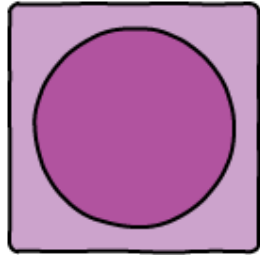
Background Color



```
background(255, 255, 255);  
fill (237, 31, 36);  
ellipse(50, 50, 80, 80);
```



```
background(118, 207, 228);  
fill (63, 75, 160);  
ellipse(50, 50, 80, 80);
```

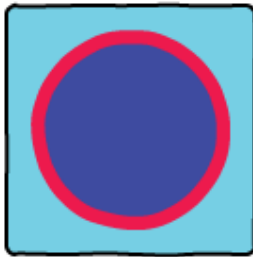


```
background(204, 163, 204);  
fill (117, 83, 160);  
ellipse(50, 50, 80, 80);
```

Stroke Color



```
background(255, 255, 255);  
noStroke();  
fill (237, 31, 36);  
ellipse(50, 50, 80, 80);
```



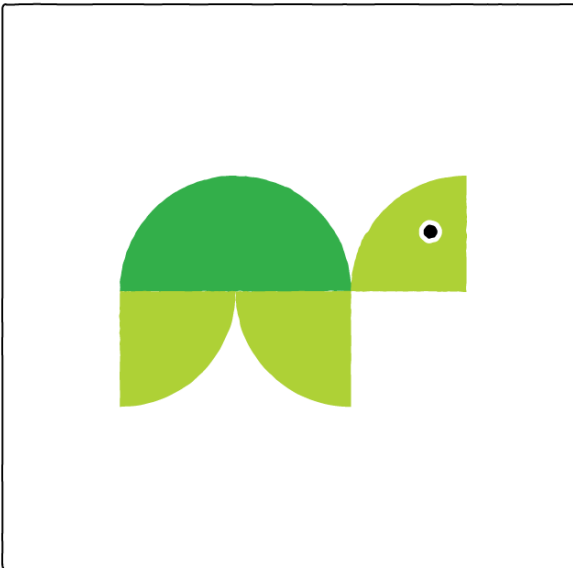
```
background(118, 207, 228);  
strokeWeight(5);  
stroke(237, 28, 78);  
fill (63, 75, 160);  
ellipse(50, 50, 80, 80);
```



```
background(204, 163, 204);  
strokeWeight(10);  
stroke(118, 207, 228);  
fill (117, 83, 160);  
ellipse(50, 50, 80, 80);
```

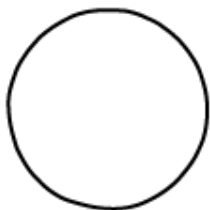
Example: Color a Turtle

```
size(250, 250);  
background(255, 255, 255); // Make background white  
  
// The turtle is created out of 4 arcs  
noStroke();  
fill(52, 174, 74); // Dark green for shell  
arc(100, 125, 100, 100, PI, (PI * 2)); // Shell  
fill(174, 209, 54); // Light green for legs and head  
arc(50, 125, 100, 100, 0, (PI / 2)); // Left leg  
arc(150, 125, 100, 100, (PI / 2), PI); // Right leg  
arc(200, 125, 100, 100, PI, (PI / 2) * 3); // Head  
  
// The turtle's eye is an ellipse  
strokeWeight(3); // Add a line around the eye  
stroke(255, 255, 255); // Make the eye line white  
fill(0, 0, 0); // Black pupil  
ellipse(185, 100, 7, 7); // Eye
```



Practice Problems

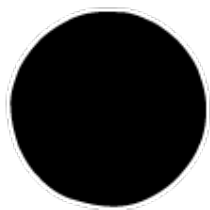
1) Find the RGB values for the colors white and black.



R _____

G _____

B _____

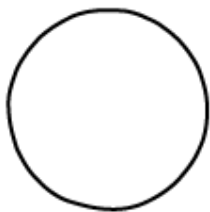


R _____

G _____

B _____

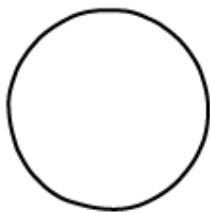
2) Find the RGB values for 3 of your favorite colors. Use a marker or color pencil to fill in the circle with the color.



R _____

G _____

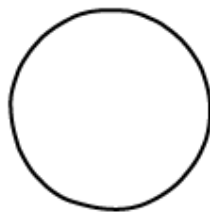
B _____



R _____

G _____

B _____



R _____

G _____

B _____

3) Using the `background()`, `fill()`, and `stroke()` functions, color the butterfly.

```
// Set the size of the display window  
size(250, 250);
```

```
//Butterfly's wings
```

```
arc(125, 125, 150, 150, -(PI/4), (2 * PI)/6);  
arc(125, 125, 150, 150, (3 * PI) / 4, (8 * PI) / 6);
```

```
// Spots on butterfly's left wing
```

```
ellipse(80, 100, 20, 20);  
ellipse(100, 120, 15, 15);  
ellipse(70, 140, 20, 20);
```

```
// Spots on butterfly's right wing
```

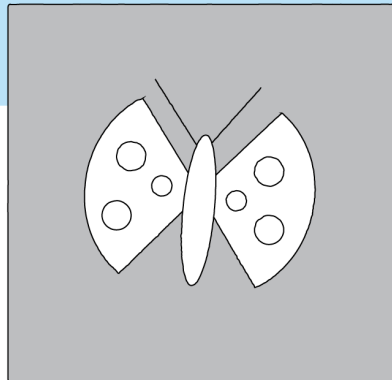
```
ellipse(170, 110, 20, 20);  
ellipse(150, 130, 15, 15);  
ellipse(170, 150, 20, 20);
```

```
//Butterfly's torso
```

```
rotate(radians(5)); // Rotate torso by 5° clockwise  
translate(10, 0); // Move torso to the right by 10 pixels  
ellipse(125, 125, 20, 100);
```

```
//Butterfly's antennas
```

```
line(120, 80, 90, 40);  
line(130, 80, 160, 40);
```



Variables

06

What is a variable? If we were programming a video game, we would want to keep track of the score as the player gains and loses points. In order to do that, we would need to store the current score in the computer's memory. **Variables** are a way for programmers to store a single value in a program.

How to Create a Variable When we create a variable, we are creating a storage space in memory to store a single value. To create a variable, we have to assign the variable a data type, a name, and a value. Storing a value in a variable for the first time is called **initializing the variable**.

```
int myFirstVariable = 5;
```

data type variable name value

Data Type The data type is the kind of value we are going to store in the variable. The chart on the next page lists the common data types. For example, if we were going to store a decimal number in the variable, we would use a float data type.

Common Data Types		
Data Type	Description	Examples of values
int	whole number	-2, -1, 0, 1, 2, 3, 4, 5
float	decimal number	-1.203, 5.67, 90.3, 121.0
String	a word	"banana," "green," "five"
char	a single character	'c,' 'w,' 'A,' 'R,' '8'
boolean	true or false	true, false
color	a color	color(5, 4, 67), color(12, 0, 111)

Variable Name Programs have lots of variables, so we need a way to tell the variables apart. The way we do this is by assigning the variable a name. There are a few rules you need to observe when making a variable name.

- 1) Always begin your variable name with a letter.
- 2) There should be no spaces in the name. Programmers use **camel case** to make variable names easier to read. The first word is lowercase and each successive word has its first letter capitalized (for example: myFirstVariable, aVeryLongVariableName, redColor).
- 3) Variable names are case sensitive, so the variable name finalScore is not the same as finalscore.

Value On the right side of the equal sign you assign the value to be stored in the variable. In your program, this value can be changed as long as the data type of the new value is the same as the data type of the old value.

A Simple Program

Lets run through a simple program that will swap the values held in two variables called “firstValue” and “secondValue.” As you follow along, it is helpful to think of variables as boxes that can store only one value. These boxes also have photocopyers to make copies of values and shredders to destroy values.

```
// Declare and initialize the variables
```

```
int firstValue = 5;
```

```
int secondValue = 17;
```

```
int temporary = 0;
```

```
// Swap the values in firstValue and secondValue
```

```
temporary = firstValue;
```

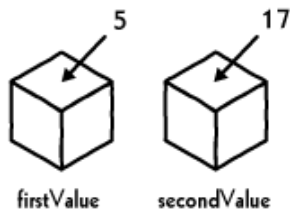
```
firstValue = secondValue;
```

```
secondValue = temporary;
```

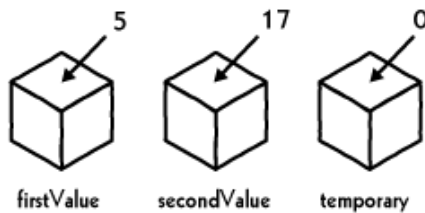
Step 1) On the first line of the program, we declared a variable with the name `firstValue` and initialized it with the value 5. When we create a variable, we are making space in memory for the variable to be stored. A variable only needs to be declared once.



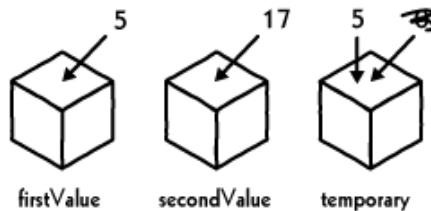
Step 2) On the second line of the program, we declared another variable, this one with the name `secondValue` and initialized it with the value 17.



Step 3) We declared a third variable on the third line of the program called **temporary** and initialized it with the value 0. I've called this third variable "temporary" because its only purpose is to help us swap the values in "firstValue" and "secondValue."

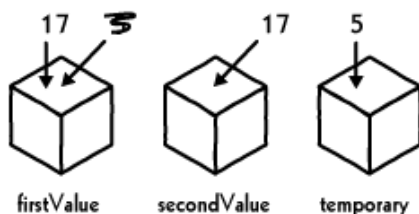


Step 4) We've declared all our variables! Now we can start swapping the values. In the fourth line of the program we see **temporary = firstValue**. This means that we are going to replace the value of 0 in "temporary" with a copy of the value in "firstValue"

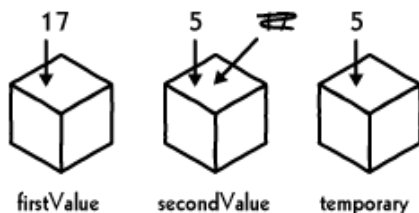


When we do this, the previous value held in "temporary," in this case 0, is destroyed and can never be accessed again.

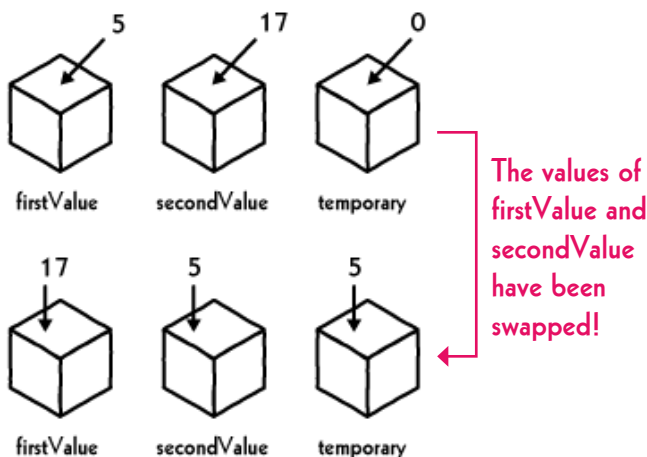
Step 5) Next, we see that `firstValue = secondValue`. This means that we are going to replace the value in “firstValue” with a copy of the value in “secondValue.”



Step 6) Finally, on the last line of the program, we see that `secondValue = temporary`.



Step 7) If we compare the first and last steps of the program, we can see that the values of “firstValue” and “secondValue” have been swapped.



Practice Problems

- 1) True or false. Are these two variable names the same: redColor and RedColor?
- 2) What are the final values stored in x and y after the program executes?

```
int x = 7;  
int y = 2;  
x = y;  
y = 3;
```

The final value of x is: _____
The final value of y is: _____

- 3) What are the final values stored in red and blue after the program executes?

```
String red = "red";  
String yellow = "yellow";  
red = "blue";
```

The final value of red is: _____
The final value of yellow is: _____

Hint: If you are struggling to keep track of the variable values in your mind, it is extremely helpful to draw a picture to keep track of the values held by the variables as they are copied and destroyed.

Printing Values

07

In the previous program, we use pictures to demonstrate what was happening to the values stored in two variables as we swapped their values. If we want to see what happens to the values as a computer runs through the program, we can use a function called `print()`.

print() Function The `print()` function writes text to the console area, which is a black box under the text editor (see page 6 for an image). Type the following into the text editor and then click run.

```
// Declare and initialize the variables
int firstValue = 5;
int secondValue = 17;
int temporary = 0;

// Print the values in the variables before swapping
print("Before swapping\n");
print("firstValue is: " + firstValue + " and secondValue
    is: " + secondValue + "\n");

// Swap the values in firstValue and secondValue
temporary = firstValue;
```

```
firstValue = secondValue;  
secondValue = temporary;
```

```
//Print the values in the variables after swapping  
print("After swapping\n");  
print("firstValue is: " + firstValue + " and secondValue  
is: " + secondValue + "\n");
```

The program should print in the console:

```
Before swapping  
firstValue is 5 and second Value is 17  
After swapping  
firstValue is 17 and secondvalue is 5
```

If you looked at the example above, you probably saw a few things that confused you, like quotation marks around certain words, the use of “\n,” and the use of the addition (+) operator.

All About Strings In processing, words are called Strings (with an uppercase “S”). You can tell the difference between a String and a variable name because a String has a quotation marks around the word and a variable name has no quotation marks around the word.

```
String colorOne = “yellow”;
```

Creating a New Line When you are using a word processor, if you want to start a new line of text, you hit the “return” key on the keyboard. To start a new line of text

while using the `print()` function, you need to use the two characters “\n” together with no spaces between the backslash and the letter n. This tells the computer that any text after “\n” needs to be on a new line.

```
print("This sentence is on one line\n");  
print("This sentence is on the next line");
```

```
This sentence is on one line  
This sentence is on the next line
```

println() Function The `println()` function is exactly the same as the `print()` function except it adds a newline to the end of the output.

```
println("This sentence is on the first line");  
print("This sentence is on the second line ");  
println("This sentence is not on a new line");  
print("This sentence is on the third line");
```

```
This sentence is on the first line  
This sentence is on the second line This sentence is not on a new line  
This sentence is on the third line
```

Concatenating Text When you see the plus sign (+) in the print() function, it means something different than what you are used to in math class. The plus sign tells the computer to **concatenate**, or combine, two strings.

```
String firstName = "George"  
String lastName = "Washington"  
print("The president of the US is:" + firstName +  
      secondName);
```

If you ran the previous example in Processing, you probably noticed that there were no spaces between some of the words. We will have to add our own spaces.

```
String firstName = "George"  
String lastName = "Washington"  
print("The president of the US is: " + firstName +  
      " " + secondName);
```

add spaces in
these two places

The president of the US is: George Washington

Doing Math with Variables

08

Addition, subtraction, multiplication and division are some of the most basic tasks you will perform as a programmer. In fact, doing math is so common in programming that shortcuts have been developed to represent certain mathematical operations.

Basic Math Operators		
Operator	Description	Example
+	addition	$5 + 3$
-	subtraction	$4 - 3$
*	multiplication	$8 * 2$
/	division	$4 / 2$

Basic Math Below is a simple example of how to add two integer numbers. At the end of the program, value printed out should be 8.


```
int numberOne = 5;
int numberTwo = 3;
int sum = numberOne + numberTwo;
print("Sum is: " + sum);
```

Assignment Operators You can also do math as part of assigning a value to a variable. For example, if you wanted to add five to a value held by a variable, you can do the following:

```
int i = 4;
i = i + 5;
print(" i is: " + i);
```

The above computation takes the current value held by the variable “i,” adds 4 to it, and then puts the new value back into “i.” At the end of the program, “i” should be 9. This type of computation is so common that a shortcut was developed to make it less cumbersome for programmers to write.

Assignment Operators		
Operator	Same as	Example
+=	i = i + 4	i += 3
-=	i = i - 2	i -= 5
*=	i = i * 5	i *= 6
/=	i = i / 2	i /= 2

Increment and Decrement Operators The most common computation that programmers do is adding 1 and subtracting 1 from a value. In fact, this operation is so common that there are special operators called the increment (++) and decrement operators(--) to make adding 1 and subtracting 1 easier to write.

Incrementing For example, the following program adds 1 to 6, and should print 7.

```
int number = 6;
number++;
print(number is: " + number);
```

Decrementing The next program subtracts 1 from 6, and should print 5.

```
int number = 6;
number--;
print(number is: " + number);
```

Increment and Decrement Operators

Operator	Same as	Example
++	i = i + 1	i++
--	i = i - 1	i--

Practice Problems

1) What is the final value of sum in this program?

```
int x = 5;  
int y = 10;  
int sum = x + y;  
print ("Sum is: " + sum);
```

2) What is the final value of x in this program?

```
int x = 3;  
x += 5;  
print("x is: " + x);
```

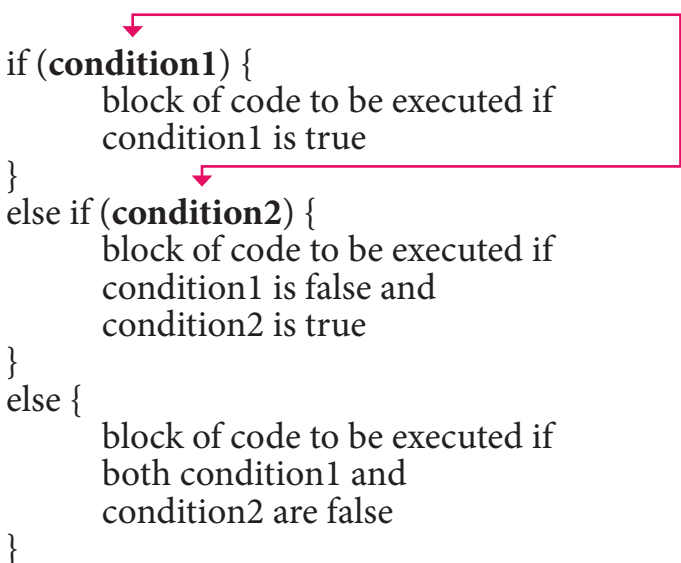
3) What is the final value of y in this program?

```
int y = 10;  
y -= 2;  
y++;
```

If / Else Statements

09

What if you wanted to award points to a player in a game based on how many gold tokens they collected. If they got less than 3 tokens, they will only get 5 extra points, but if they got 3 or more tokens then they will get 10 extra points. An easy way to program this is to use an if / else statement.



```
if (condition1) {  
    block of code to be executed if  
    condition1 is true  
}  
else if (condition2) {  
    block of code to be executed if  
    condition1 is false and  
    condition2 is true  
}  
else {  
    block of code to be executed if  
    both condition1 and  
    condition2 are false  
}
```

Conditions
must be
boolean
statements

The if / else statement tells your program to execute a certain section of code only if a particular condition is true. A condition must be a **boolean** statement, which means that it must evaluate to either true or false.

Relational Operators In order to construct a boolean statement, you must compare two values with a **relational operator**. A relational operator determines if one value is equal to, not equal to, greater than, less than, greater than or equal to, or less than or equal to the other value.

Relational Operators		
Operator	Meaning	Example
==	is equal to	a = b
!=	is not equal to	a != b
<	less than	a < b
>	greater than	a > b
<=	less than or equal to	a <= b
>=	greater than or equal to	a >= b

The Double Equal Sign You many have noticed in the above table that when you compare two values to see if they are equal, you must use a double equal sign (==). In Processing, a single equal sign is used to assign a value to a variable and two equal signs is used as a test for equality.

It is easy get the two confused (even experienced programmers make this mistake all the time), but just remember that whenever you are *comparing* two things to see if they are equal, to use a double equal sign.

Example If number of tokens collected is less than 3, increase the score by 5 points. If number of tokens collected is greater than or equal to 3, increase the score by 10 points.

```
// Declare and initialize variables
int score = 30;
int tokens = 3;

// Start if / else statement
if (tokens < 3) {
    score += 5;
    println("Your new score is: " + score + " points");
}
else {
    score += 10;
    println("Your new score is: " + score + " points");
}
```

When the program finishes, the statement printed will be "Your new score is: 40 points."

Hint: You may have noticed that there are no semicolons at the end of an if /else statement. In general, you should not put a semicolon after a closing curly bracket (}).

Practice Problems

1) What will be the final score when the program finishes running?

```
int score = 10;
if (score < 5) {
    score += 2;
}
else {
    score += 10;
}
```

2) When the program finishes, which statement will be printed?

```
int age = 13;
if (age < 13) {
    print("I am less than 13 years old");
}
else if (age == 13) {
    print("I am 13 years old");
}
else {
    print("I am older than 13 years old");
}
```

3) What if we changed the age from 13 to 16 in the above program? What statement will be printed?

While Loops

10

If you wanted to draw 10 identical ellipses in a row, you could write 10 ellipse functions like so:

```
size(500, 300);

//Declare and initialize variables
int xPos = 30;
int yPos = 30;
int diameter = 40;
int space = 49;

// Draw 10 circles
ellipse(xPos, yPos, diameter, diameter);
ellipse(xPos + space, yPos, diameter, diameter);
ellipse(xPos + 2 * space, yPos, diameter, diameter);
ellipse(xPos + 3 * space, yPos, diameter, diameter);
ellipse(xPos + 4 * space, yPos, diameter, diameter);
ellipse(xPos + 5 * space, yPos, diameter, diameter);
ellipse(xPos + 6 * space, yPos, diameter, diameter);
ellipse(xPos + 7 * space, yPos, diameter, diameter);
ellipse(xPos + 8 * space, yPos, diameter, diameter);
ellipse(xPos + 9 * space, yPos, diameter, diameter);
```

Writing out 10 ellipse functions is not very hard, but what if you wanted to draw 200 ellipses or 1000 ellipses? It would be a lot of work to write all those ellipse functions (and even more work if you make a mistake in your cal-

culations and have to correct every ellipse function).

Luckily for us, there is something called a **while loop** that allows us to draw as many ellipses as we want in just a few lines of code.

```
int counter = 0;
while (loop condition) {
    While the condition is true,
    execute this block of code
    Increment counter by 1 (usually)
}
```

The loop condition must be a boolean statement

Loop Body

While Loop Formatting The text between the parentheses is called the **loop condition**. The loop condition must always be a boolean, which means that the loop condition must be true or false (see page 37 to learn about making boolean statements). The code between the curly brackets ({ }) is called the **loop body**. The loop body is what gets executed if the loop condition is true.

While Loop Example

```
int counter = 0;
int i = 0;
while(counter < 10) {
    i += 5;
    counter++;
}
```

How the While Loop Works When Processing gets to a while loop in a program, it first checks the loop con-

ditional (in the example above, the loop conditional is `counter < 10`) to see if the statement is true. If it is true, it executes all the code in the loop body.

When Processing reaches the end bracket, it checks again to see if the conditional (`counter < 10`) is still true. If it is, Processing jumps back to the beginning bracket and processes all the code in the loop body.

As long as the conditional is true, Processing will continue to execute the code in the loop body. Processing can not exit the loop until the conditional is false.

Step-By-Step While loops can be difficult to understand at first. It can be very helpful to walk through the code yourself line by line in order to understand what Processing is doing when it executes a while loop.

```
int counter = 8;
while (counter < 12) {
    println("counter is: " + counter);
    counter++;
}
println("counter is greater than or equal to 12, exit the
       while loop");
```

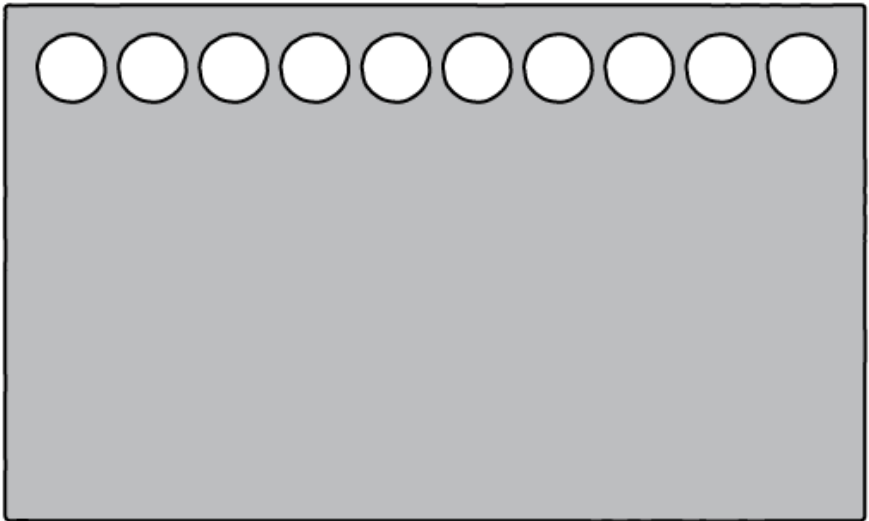
```
counter is: 8
counter is: 9
counter is: 10
counter is: 11
counter is greater than or equal to 12, exit the while loop
```

Example: Drawing Circles

```
size(500, 300);

// Declare and initialize variables
int counter = 0;
int xPos = 30;

// Draw 10 identical circles in a row
while (counter < 10) {
    // Draw the ellipse
    ellipse(xPos, 30, 40, 40);
    // Shift the x-coordinate of the center of the circle
    // over by 49 pixels
    xPos += 49;
    // Increment the counter by 1
    counter++;
}
```



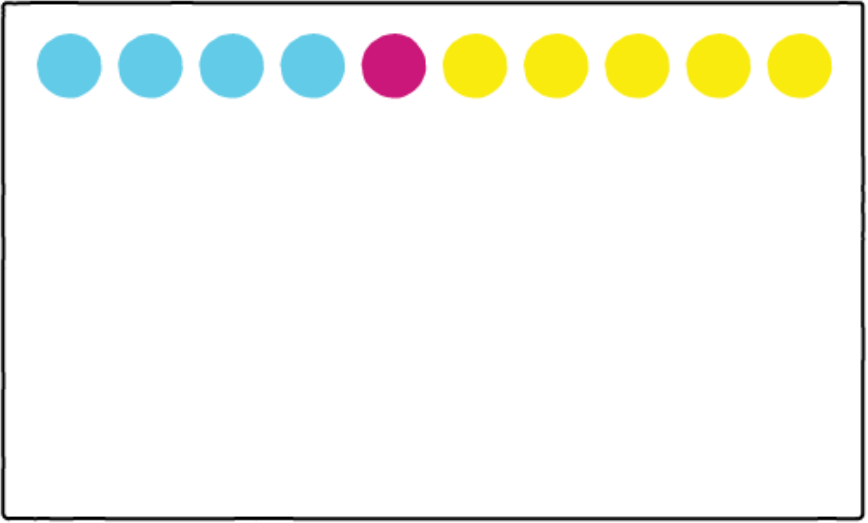
Example: Coloring Circles

```
size(500, 300);

// Declare and initialize variables
int counter = 0;
int xPos = 30;

// Set the background color and remove the stroke
background(255, 255, 255);
noStroke();

// Draw and color the circles
while (counter < 10) {
    // Make the first 3 circles blue
    if (counter < 4) {
        fill(25, 233, 255);
        ellipse(xPos, 30, 40, 40);
    }
    // Make the 4th circle pink
    else if (counter == 4) {
        fill(204, 20, 121);
        ellipse(xPos, 30, 40, 40);
    }
    // Make remaining 5 circles yellow
    else {
        fill(255, 233, 0);
        ellipse(xPos, 30, 40, 40);
    }
    // Shift the x-coordinate of the center of the circle
    xPos += 49;
    // Increment counter by 1
    counter++;
}
```



Can you think of any other order you would want to color the circles? What if you wanted to color every other circle a different color?

Practice Problems

1)

Name:

Email:

Text and Illustrations By: Nicole Yarroch