VARIABLES

In p5.js, we can think of a **variable** as a container for a value. That value can be a number, an image, a file, etc. A variable has a unique name, and, every time you "call" that name, the computer program will replace it with whatever value is stored there.

-

There are several types of variables. A **global variable** can be called at any point during your sketch. These are placed at the top of the sketch, before the setup, and you can refer to that variable at any point during the sketch. Global variables also give you quick access to certain parameters within your sketch.

For example, you may have a circle (or many) in your sketch, and you may want to have easy access to quickly change the diameter. An easy way to do this would be to use a variable in place of the height and width parameters in your ellipse. In this case, we'll call our variable "d" for diameter. So,

```
ellipse(600, 400, 100, 100);
```

becomes

```
ellipse(600, 400, d, d);
```

And at the very top of the sketch, before the setup, we'll define the variable as such:

```
var d = 100;
```

Now everytime I use the letter d, the value "100" is used instead. If I want to quickly change the size of the circle to be 200 pixels in diameter, I simply replace the value at the top of the sketch.

-

A **local variable** is created for a specific purpose, and then discarded. The program does not remember that variable after it has been utilized. We will use them a little later, in the iteration section.

p5.js also has a number of **built-in variables**. These are variables that every program is keeping track of anyway. Some examples of built-in variables are mouseX and mouseY, which hold the values of your mouse's x and y position at any point in time, or width and height, which hold the value in pixels of the width and height of your canvas.

ITERATION

If you find yourself typing or copying the same lines of code over and over again in a sketch, or if you notice a mathematical pattern emerging, it is very likely that there will be a shorter way to write that code. While computer programs are not very good at things like empathy and funny jokes, they *are* very good at doing the same thing over and over again with precision.

A common and powerful way to code for repetition is using a **for loop**. A **for loop** uses a **local variable** to repeat something over and over again making a steady change each time. For example, if you are drawing three ellipses in a row, you might write something like this (note that the d is a global variable, from the example above).

```
ellipse(400, 400, d, d);
ellipse(600, 400, d, d);
ellipse(800, 400, d, d);
```

We can see that the majority of these lines are repeated, with only the first parameter (the x position), changing. This is the perfect opportunity to make a **for loop** that repeats most of this line, while iterating through a set of three variables, that being at 400 and end at 800, and move incrementally in steps of 200. To do that, we would use the following construction:

```
for(var i = 400; i <= 800; i = i+200){
      ellipse(i,400, d, d);
      }
```

The lines inside of the brackets { } will run, for as long as the conditions inside the parenthesis after "for" are met.

To break it down:

```
var i = 400; //This establishes the local variable, "i", and gives it an initial
             value, in this case, a value of 400. The first time this loop runs, the
             "i" will be replaced with the value of 400.

i <= 800;    //This establishes the condition our variable must meet to continue
             running the for loop. In this case, we want the loop to run for as long
             as "i" is less than or equal to 800.

i = i +200;  // This happens everytime, and establishes how our variable changes after
             every time the loop is run. Here, we want our "i" value to increase by
             200 each time.
```

To sum it up, we want to create a variable "i," that has an initial value of 400, which increases by 200 each time we run the loop, and only for as long as the variable is less than or equal to 800. So we run this code three times - the first replaces "i" with 400, the second time with "600", the third with "800."

Try generating hundreds of circles, or "nesting" for loops by creating multiple local variables, to witness the power of repetition with a for loop.

INTERACTIVITY

Because the p5.js fits so easily into a web browser, you may want to incorporate different ways of working with input from your viewer. The most direct forms of user input involve the mouse and the keyboard. There is a great tutorial on interactivity on p5.js, but I will highlight some of the most immediately useful techniques.

Mouse Data

```
mouseX //Stores the x position of the cursor (from the top left side the canvas)
mouseY //Stores the Y position of the cursor (from the top of the canvas)
```

To print the mouse positions to your console, include the following line in your draw loop:

```
print(mouseX, mouseY);
```

Then to open your console, Press **Command+Option+J (Mac)** or **Control+Shift+J (Windows, Linux, Chrome OS).** You will see the

mouseX and mouseY can also be used as variables in any parameter in your sketch. Try replacing them with the position of an ellipse, or, alter the color of the background as you move your mouse around the screen.

```
pmouseX //previous x position of cursor
pmouseY //previous y position of cursor
```

Mouse Buttons

```
mouseButton      //variable that returns LEFT, CENTER, or RIGHT depending on the mouse button
                 most recently pressed. Retains its value until a different button is pressed.
mouseIsPressed   //variable that returns true if any mouse button is pressed and false if no
                 mouse button is pressed. Reverts to false as soon as the button is released.
```

Mouse Events

```
mousePressed()   //Code inside this block is run one time when a mouse button is pressed
mouseReleased()  //..runs one time when a mouse button is released
mouseClicked()   //..runs once after a mouse button is pressed and released over the element
doubleClicked()  //runs once after a mouse button is pressed and released over the element twice
mouseMoved()     //runs one time when the mouse is moved
mouseDragged()   //runs one time when the mouse is moved while a mouse button is pressed
mouseOver()      //runs once after every time a mouse moves onto the element.
mouseOut()       //runs once after every time a mouse moves off the element
```

Keyboard Data

```
keyIsPressed     //Boolean variable that returns TRUE if a key is pressed and FALSE if not. Used
                 with “if” statements to set up code that runs when keys are pressed.
key              //variable that stores the most recently pressed alphanumeric key character
keyTyped()       //runs code within the parenthesis only when a key is typed - can be used to
                 change values of other variables.
```

CONDITIONALS & LOGICAL OPERATORS

Conditionals allow you to set up certain conditions under which code is executed. The **for loop** established certain conditions under which the code inside the for loop would run.

A **conditional** establishes a condition, the program checks it, and if the condition is met, then the following code will run. Conditionals include **for**, **if**, **else**, and **else if**.

For example, if I want to change the background black when the mouse is on the right side of the screen, and white on the left, I would use the following code:

```
if (mouseX > 600){
 background(0);
}

else{
 background(255);
}
```

A **logical operator** allows you to combine conditionals into more complex expressions. These include: **&&** (AND) and **| |**(OR). For example, to change the background black only if the mouse is in the bottom right quadrant:

```
if ((mouseX > 600) && (mouseY > 400){
 background(0);
}

else{
 background(255);
}
```

OTHER HELPFUL FUNCTIONS

```
random(255);    //returns a random, floating-point number. One argument will choose a number
                between 0 and your specified value (in this case 255), two arguments will produce
                a random number between two specified values

map(value, 0, 800, 0, 255);
                //Re-maps a number from one range to another. First number is the incoming value
                to be converted, then the min and max of the value's current range, then the min
                and max of the converted range.

noCursor();     //makes the cursor disappear

fullscreen();   //sets your window into full screen when condition is met
```