

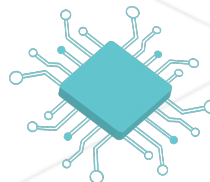


## APCSP - Part00

### Variables, Strings, Output, and Built-in Functions

Kai [kai@42.us.org](mailto:kai@42.us.org)

*Summary: Learn how the basic building blocks of p5.js are going to work.*



**HACK  
HIGH  
SCHOOL**



*This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).*

# Contents

<b>I</b>	<b>The Processing Community</b>	<b>2</b>
<b>II</b>	<b>Your First Sketch</b>	<b>3</b>
II.1	Understanding the Example Code . . . . .	4
<b>III</b>	<b>Color</b>	<b>6</b>
III.1	Homework pt. 1 . . . . .	6
<b>IV</b>	<b>Let's Say Something: Output!</b>	<b>9</b>
<b>V</b>	<b>Comments</b>	<b>10</b>
<b>VI</b>	<b>Values and Variables</b>	<b>11</b>
VI.1	Printing Variables . . . . .	11
VI.2	Things you can do with variables . . . . .	11
VI.3	Rules for naming variables . . . . .	12
<b>VII</b>	<b>Parameters</b>	<b>13</b>
<b>VIII</b>	<b>Coordinate System</b>	<b>14</b>
<b>IX</b>	<b>Let's Draw Something: Shapes!</b>	<b>15</b>
IX.1	Homework pt. 2 . . . . .	15
<b>X</b>	<b>Go Further</b>	<b>17</b>
X.1	Homework Bonus pt. 1 . . . . .	17
X.2	Homework Bonus pt. 2 . . . . .	17
<b>XI</b>	<b>Super Bonus</b>	<b>18</b>

# Chapter I

## The Processing Community

Welcome to class! In this self-guided curriculum we will be exploring the world of computers through creative coding in the p5.js graphics library.

P5.js is an implementation of Processing, a graphics library which is popular with both Computer Science and Graphic Design classes worldwide.

Go ahead and check it out: Here are some relevant websites you may want to bookmark.

Processing.org: The main website of the Processing Foundation, which created and supports the graphics library. The graphics library is originally written in Java and they have translated it into the p5.js version for Javascript programmers, processing.py version for Python programmers, and also Processing versions to create programs for Android phones and Rasberry Pi's.

p5js.org: Here is the main reference page for everything in the Javascript version of Processing! You will need to the "Reference" section on a regular basis to look up what functions and keywords are used in the language. The section under "Learn" has some helpful tutorials.

openprocessing.org is a community where you can find a lot of inspiration from other people's projects in Processing. Just explore here to see what is possible and get ideas! Be careful of two things: (1) Some of the projects will be in Java syntax, not Javascript. (2) Don't copy and paste code from here - if you copy other people's code you won't understand it and won't know how to debug!

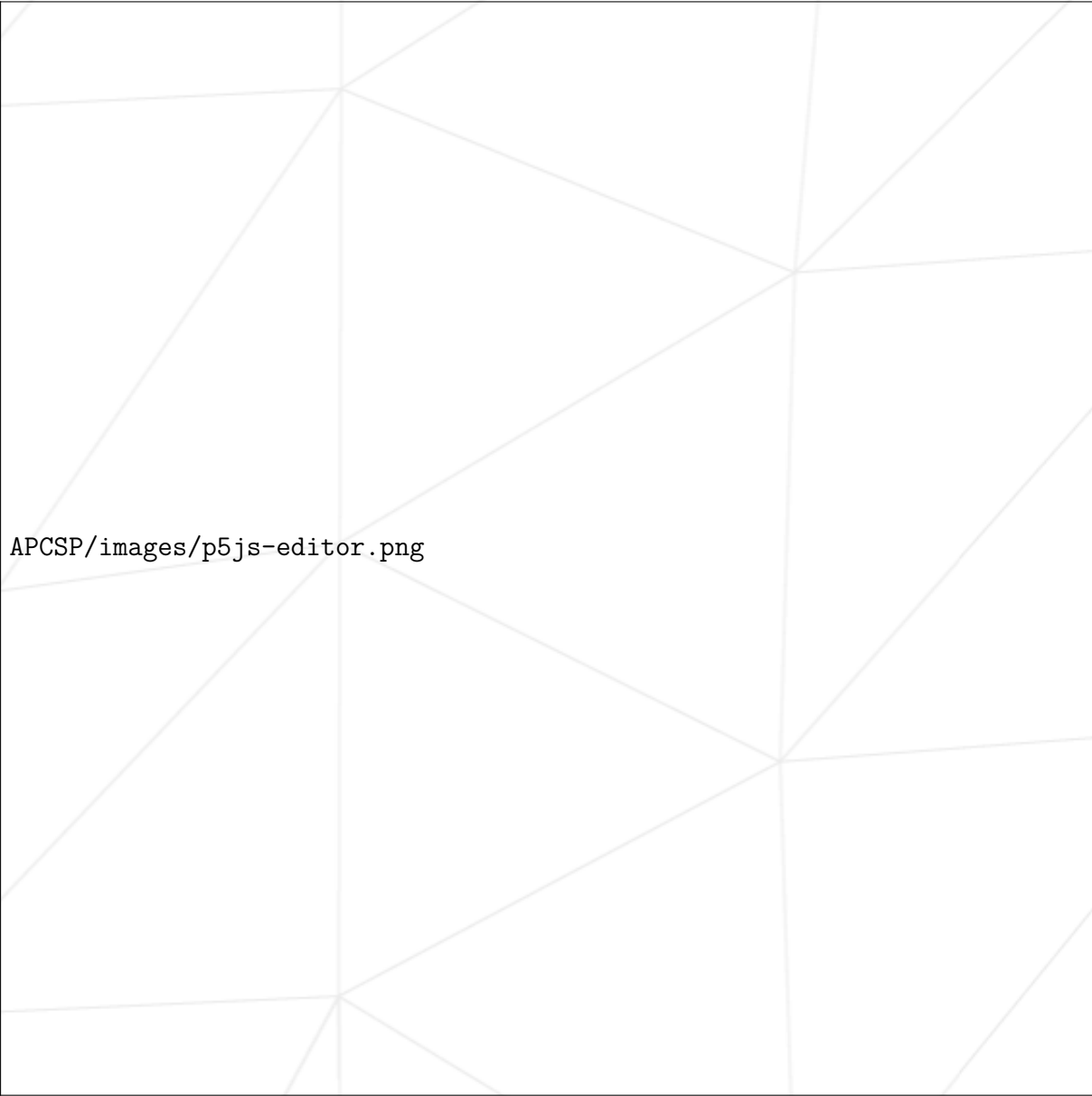
editor.p5js.org is an online code editor where you can write Processing code in your browser. Openprocessing also has one at [openprocessing.org/sketch/create](https://openprocessing.org/sketch/create). Openprocessing's editor is nice because you can follow other people's profiles; p5js's editor is more of a private creative space. Let's start with p5js first.

discourse.processing.org is the discussion forum for the Processing community. You can click on the "Categories" drop down option to filter for conversations that have to do with p5.js, instead of the other versions of Processing. Be good citizens and follow the [Community Guidelines](#) when you post here!

# Chapter II

## Your First Sketch

Make an account [editor.p5js.org](https://editor.p5js.org), and then open your first sketch.



`APCSP/images/p5js-editor.png`

Notice a few things:

- Your sketch is given a cute auto-generated title. In my case they called it "Forgoing tendency." You can click on that title to give it your own name.
- The red and white arrow is the Play button. Click it to see your animation appear on the right under Preview.
- The red square is the Stop button.
- The Console below your code is just like Terminal, on your computer. You may see error messages pop up here, and you can also print out your own notes about what is happening in the program.

## II.1 Understanding the Example Code

Let's take a look at the boilerplate code that is generated when we open a new sketch.

**function:** Anything labelled as a function is a reusable block of code that has the ability to take in inputs and return an output.

In math class we would say it like this:

$$f(x) = a + b$$

But in Javascript we say it like this:

In the "add" example, the inputs are "a" and "b", and the output is the sum.

**setup()** is the built-in function that will run every time your animation begins. Any code that you write inside setup will run exactly once after pressing start.

**CreateCanvas()** is a library function that you should always have in your setup block. It takes in two inputs: a width and a height. In the default example, width and height have both been set to 400 units. Click the Play button (forward arrow) to see what happens. Then, change the width and height values, and click play again. Do you see how they control the size and shape of the grey background?

**draw()** is the built-in function that will repeat over and over after setup() runs. If you want the animation to move and change over time, you will write code inside of draw() that makes it happen.

**background():** The default program starts with a background(220) command inside of the draw block. In this example, 220 is a **greyscale** value which defines the shade of grey that we will draw for the background.

Try changing the 220 number and see what happens.

Color values are traditionally on a spectrum from 0 to 255! So for these greyscale values, you will notice that 255 is white, and 0 makes the background black.



Why 255? Because computer science always uses numbers that are powers of 2 (2... 4... 8... 16... 64... 128... 256!) But we start counting at 0. So, for example, there are ten digits in the range from 0 to 9. And there are 256 greyscale values between 0 and 255.

# Chapter III

## Color

If you followed my link to the reference documentation on [background\(\)](#) above, then you may have noticed there are several options for how to define color in p5.js.

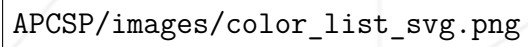
Go ahead and read through that page, and you may want to Google some of the terms if you don't know about them. There are a lot of nice online resources for understanding color.

Let me know if you find a color picker even nicer than [Colorizer.org!](#)  
Be prepared to answer these questions during corrections:

- What is the default color mode in Processing?
- What's your favorite SVG/CSS Color String name?
- What is the A in RGBA?
- What do H and S stand for?
- Do hexadecimal codes represent colors in RGB, HSV or CMYK?
- Bonus: What is CMYK?

### III.1 Homework pt. 1

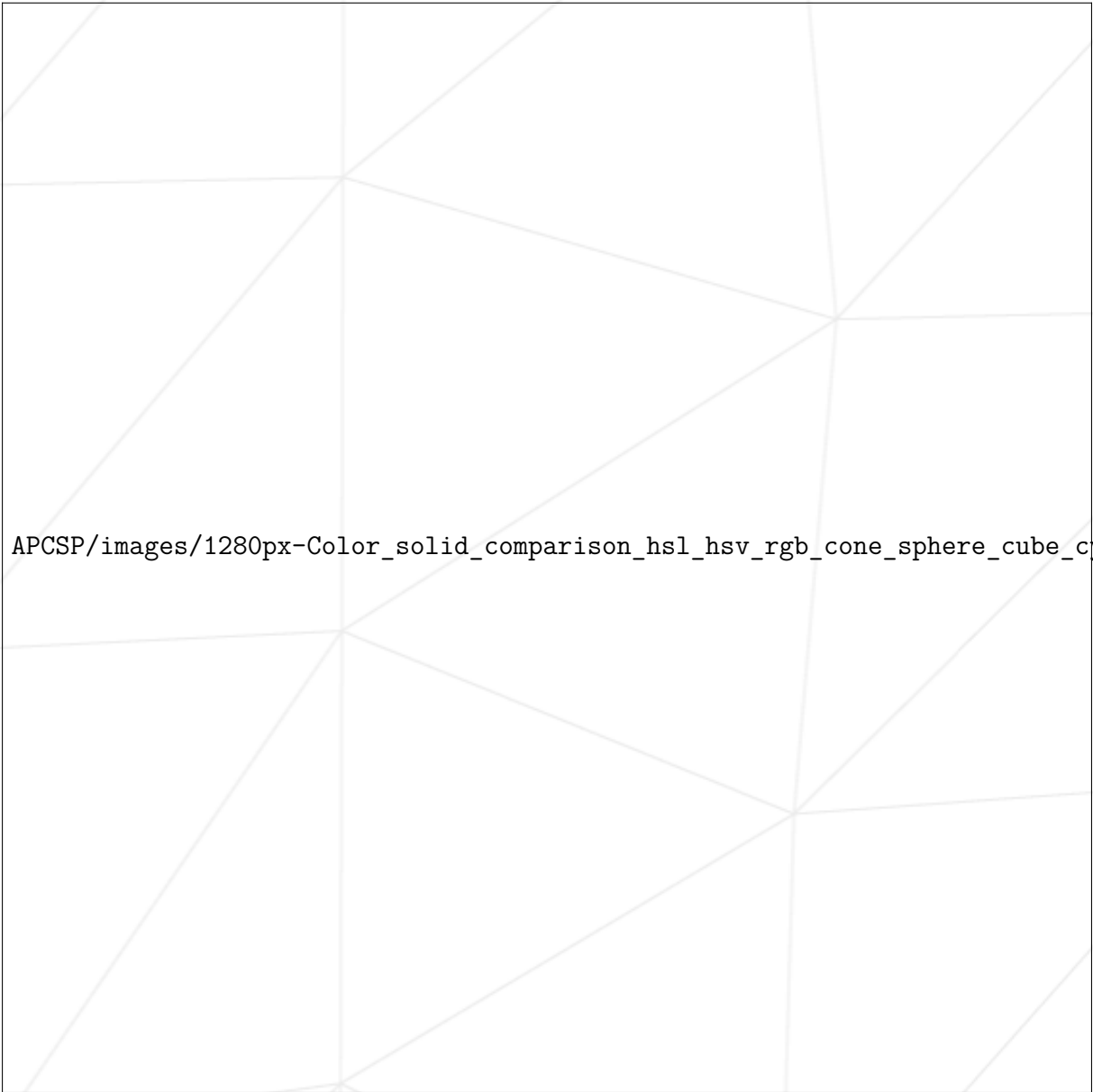
Study the questions above, and also change the background on your project to a favorite color.



APCSP/images/color\_list\_svg.png

- image from [GLE](#)





- image from [Wikipedia](#)

# Chapter IV

## Let's Say Something: Output!

Coding is often a process of trial and error, where you write some code the way you think is correct, and then run it to see what happens. When it runs it may be perfect (Yay!), or you may need to analyze the output logically to **debug** the program.

The most straightforward way to debug your program is to add print statements which help you identify which steps are happening in what order as the program runs. The Processing library adds a `print()` statement which you can use for this purpose.

Here's an example:

Add these print statements to the code in your p5js editor, and run the program. Do you see how the output of the print statements appears in the Console section below your code?

Here are a few things to know about `print()`:

- You can print literal text, or variables. If you are printing literal text it should be inside either 'single' or "double" quotes.
- `Print()` is not the text output function used by the rest of the Javascript community - they would use `alert()` to create a popup, or `console.log()` to print to console. Processing has added `print()` as a nice synonym for `console.log()`.

# Chapter V

## Comments

The other great way to keep track of what is happening in your code is to add **comments**.

Any line that begins with two forward slashes (//) will be ignored by the computer - it does not count as code.

You can use that to take notes to remind yourself of what a line of code does.

You can also use it to temporarily remove lines of code without erasing them.

# Chapter VI

## Values and Variables

A **value** is any single piece of data that is stored inside code. It could be a string or a number, or other things. We can use values directly, or store them inside a variable.

**Strings** are text values such as are words, sentences, letters or paragraphs. You tell the computer it's a string by wrapping it in double or single quotes. "42" is a string. So is "42 is totally cool."

**Numbers** are numeric values. 42 is a number if you don't put quotation marks around it, and so is 42.42. You can use numbers for math operations, which do not really work on strings.

A **variable** is like a box that has a label on the outside describing its contents, and can store one value at a time.

### VI.1 Printing Variables

You can embed variables inside of strings when you use the `print()` statement, and it is very useful for debugging. Here are two ways to do it:

Notice that in the second print statement example, I surround the content not with quotation marks but with **backticks**. It's the symbol in the top left of your keyboard on the same key as the tilde (~).

### VI.2 Things you can do with variables

- **Declare:** You can create an empty box and label it, with the syntax `"var myvariable"`. The variable then has no value (it is unassigned) after declaration. You'll get an **error message** which points this out to you if you try to print it.
- **Assign:** You can place a value inside the box with the syntax `"myvariable = 42"`. You can also declare and assign a variable on the same line.

- **Reassign:** You can replace the value that is assigned to the variable. The variable will not remember what was there before.
- **Copy:** You can copy the contents of one variable into another one. It creates a copy of the value - it doesn't make the two variables linked forever. After the example below, `myvar_b` will contain "red", while `myvar_a` and `myvar_c` will both contain "orange".

## VI.3 Rules for naming variables

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with \$ and \_ (but those have special meanings, so don't use them yet.)
- Names are case sensitive (i.e. `y` and `Y` are different variables.)
- Reserved words (like JavaScript keywords) cannot be used as names.

`var` is an example of a Javascript Reserved Word. It has special meaning in the Javascript language, so you can't use it as a variable name. [Every language has its own keywords - here's some of ours!](#)

# Chapter VII

## Parameters

We talked on page 4 about **functions**. The next word to learn is **parameter**, which is the **input(s)** that we give to a function.

When we changed the background color earlier, we were giving the `background()` function different parameters.

Parameters allow us to use one function in different ways. The `background()` function responds differently depending on whether you give it one number (output is greyscale) or three numbers (output is colorful).

In our previous example of a simple "add" function on page 4, `a` and `b` are the parameters. When you **call** the function (AKA use it), you would give real values for `a` and `b`.

You could save the `a` and `b` values as variables first, and then pass those into the function. You can also save the result as another variable:

In the example above, the values of 2 and 4 are stored in variables called `"first_number"` and `"second_number"`; but once they are sent inside the `add` function, that function calls them by the names `"a"` and `"b"`.

# Chapter VIII

## Coordinate System

Working with X and Y values in computer drawing is a little different than what you're used to in math class. In computer science, generally (0,0) is at the top left of the screen, so we are working in Quadrant IV of the Coordinate Plane.

Except, Y-values INCREASE as we move down from the top of the screen, instead of becoming negative.



APCSP/images/processing\_grid.png

Image from [Processing.org](https://processing.org)

# Chapter IX

## Let's Draw Something: Shapes!

Check out the [p5js documentation for the Shapes section](#) and play around with drawing ovals, rectangles, triangles, curves and lines to your screen. When you add the command `fill(color)` before drawing a shape, it controls what color they will be.

### IX.1 Homework pt. 2

In the picture here I have measured the dimensions of the 42 logo:





Write a Processing program that saves a variable for each x and y value that I labelled as (xa, xb, xc.. ya, yb, yc...). Then, use rectangles, quadrilaterals and triangles to draw a black 42 logo on your screen. All the parameters should be given in the form of variables that you saved. The 42 logo should be drawn in black, but the background can be any color!

# Chapter X

## Go Further

OK, are you done - was that part easy for you?  
Here's the bonus for today:

### X.1 Homework Bonus pt. 1

Read the p5js reference section on "Typography" and use what you learn to add a title to your drawing. Display the title in medium-sized white text at the top center of the canvas.



You may see that the text looks wierd or blurry if it is drawn over and over again in the same spot. If you draw the text during `setup()` instead of `draw()`, it will only be drawn one time.

### X.2 Homework Bonus pt. 2

Change the 42 logo so that it starts in the top left corner of the screen and moves diagonally down and to the right.



You'll need an x-offset and y-offset variable that change over time and which are used to modify each point in the shape. In Geometry class your teacher would call this "translation".

# Chapter XI

## Super Bonus

Done this all before?

If you know all the basics of programming already, start yourself on on The Coding Train series about Object-Oriented Programming in ES6 Javascript. [The Coding Train: Classes in Javascript with ES6](#)

Add to your sketch a Dot class which includes variables for color, x coordinate, y coordinate, and diameter. Update the x and y coordinates during setup() so that the dot moves accross the screen either in a straight line, or in a random wandering path.