



Learning to code by drawing Instructor's notes

Tutorial created by: Maria Dermentzi
@mdermentzi

Introduction

p5.js is a JavaScript library that was created to make programming more accessible to non-programmers like artists, designers, teachers, and beginners in general. It is kind of like an extension to JavaScript, to put it quite simply. By using p5.js, we turn our web browser into a canvas onto which we can draw, but also create animations and add interactivity to our “sketch.”

1) JavaScript, along with HTML and CSS, is one of the main languages that are being used to create web pages. We use JavaScript to add interactivity on static HTML and CSS content. In this course, you are provided with a folder that contains an HTML file called “index.html,” which is necessary to “load” p5.js on our website. In the same folder, you’ll also find a document called “sketch” which is where we are going to develop our program. To be able to see the results of the code you’ll be writing in “sketch.js,” you need to open the index.html file with Google Chrome by Right Clicking -> Open With -> Google Chrome.

2) At the same time, you need to open the “sketch.js” file with the help of a text editor. In this case, we’ll be using the “Atom” text editor, so you have to Right Click on “sketch.js” and select Open With-> Atom.

3) Resize those two windows so that they both fit on your screens.

Attention 1! Whenever we make a change to our code, we do not forget to save Ctrl + S / Cmd + S or File -> Save. To see the changes we've made, go to the browser window and refresh (Ctrl + R / Cmd + R) or by clicking on the refresh button that looks like the recycling sign).

Attention 2!

You will soon see that we will often need to use parentheses or square brackets in our code. When we add an open parenthesis or bracket, we need to be careful not to forget to add a close parenthesis or bracket. Also, at the end of each command, we write within the parentheses and brackets area, we need to add a semicolon.

4) Let’s have a look at the code in our Atom window. We can see that at the moment it's full of some text in italics, but there are also a few other lines of code in that are colored instead of just gray. The lines that are in gray italics are comments that programmers put in their code to remember what the code following this comment is all about, or, because comments are not executable by the browsers, sometimes they’re also used to temporarily disable a command so that we can try another one and see what works best. The web browser completely ignores the comments, so whatever we put in there will not be visible to the user of our program. The two colored pieces of code that we see, the **function setup () {}** and **function draw () {}** parts, are the two primary functions found in a sketch of p5js. The **function setup () {}** always runs first and once in our program. Inside this part of the code, we will add some basic settings of our program, which do not need to be declared a second time or repeated afterward. Instead, all of our drawings will be created in the **function draw () {}** part that runs continuously over and over again, as opposed to the function

setup () that runs only once in the beginning. We notice that this file has been divided into two regions, the upper one, in which we will define all of our variables, and the bottom one, in which we will write all of our design commands, which are further divided into a series of numbered steps that we will follow. It is critical that you do not erase these comments, because they will make it easier for you to code this program. If you delete a comment by mistake, you can undo by hitting Ctrl + Z / Cmd + Z.

5) Find **STEP 1** in your code:

The first thing we have to do is define the size of our canvas, that is, the area we want to use for our designs. For this exercise, we want to make a canvas with a width of **1024pixels** and a height of **768pixels**. To do this, we will use the **createCanvas();** command.

Once we define the size of the canvas, we can proceed to define the color that we want our background to be using **background(r,g,b)** . For the purpose of the course, all the color values we will use today are given as a comment above the line where you'll have to define the color of a shape. Please, use the suggested colors during this course, but feel free to play around with them if you finish earlier or even at home.

Note: In p5.js, all colors are defined by default as rgb values, ie we need to know how much red, how green and how much blue is contained in our desired color so that these three colors combined give us the color that we want. The r, g, b values of a color can be found through apps like Photoshop or even online by just googling for rgb colors. The values of each color can range from 0 to 255.

As we will later see, in p5.js each time we create a shape there is a black outline around. We do not want this today so we will state here in the setup function that we do not want to have stroke in around our shapes using **noStroke ();** .

6) **STEP 2:**

Let's draw! Where should we start? First, we need to think about what should be at the background and what in the foreground of our drawing. For example, does the hill appear in front of or behind the skyscrapers? We need to figure this out because the browser will start executing our code from the top of our text to the bottom and every new shape that appears later in our code will get drawn above all the previous ones.

Find the right order from the background to the foreground:
hill, stars, windows, skyscrapers, moon, sky

Before we begin to draw each shape, we will go up to the beginning of our code and define the variables of the objects we are going to design. So, let's go up to define the variable that we need for the stars and then come back down here.

NOTE: We use variables to store data such as numbers, words, etc. under a symbolic name. They are especially useful when a value is to be used several times in our program so we do not need to write it over and over again. All you have to do to reuse this data within your code is just to recall the name of the variable that you used to save it! In addition, variables are useful to make quick changes to our code.

Once we have created the variable **var starDiameter = 7;** for the diameter of the stars, we design the stars that fill the sky.

First, we choose the color with which we fill our shape with **fill (r, g, b);** .

The stars have a circular shape and for this reason, we will use the ellipse command **ellipse(x, y, horizontal diameter, vertical diameter);** . The stars must appear everywhere on the canvas in a

random position. This can be achieved with the help of **random(minValue, maxValue);** . By using **random()** we're telling the program that we want it to randomly choose a value within the framework we define. We also want our canvas to fill up with many stars, and to do this we just take advantage of the **draw()** function that, as we said, is repeated over and over again.

We start drawing the shape and declare that the location of each star on the x-axis and the y-axis will be randomly selected by the program taking values from 0 to the maximum value of the width and height of our canvas respectively. For the diameter, we use the variable we just made at the beginning of this step. Like this:

```
ellipse(random(0,width), random(0,height), starDiameter, starDiameter);
```

7) STEP 3: Let's now draw the moon. First, we choose the color with which we will fill our shape. We design the circle by finding the values we need from the reference image we were given. The x and y points in the **ellipse()** define where the center of the circle will be on the canvas. Can you guess what the x and y values we'll use in this command are?

8) STEP 4: We go back to the top of our code to create the Skyscrapers **Objects** that will help us design our shapes. What is an object?

We saw that with variables we store a value with a symbolic name. Objects are similar to variables, but they can store multiple values. Here, we want to store more than one values as properties of our variables in skysc1-skysc8 (1-8 because we're going to have 8 skyscrapers). Example: An object, for example, a cell phone, has some properties/attributes such as manufacturer, model, color, etc. These properties on a specific phone have specific values like Apple, iPhone 5s, Black. All mobile phones have the same properties but their properties do not have the same values.

The skysc1-skysc8 objects, which are about to obtain a **rectangular shape, rect()**, have the following properties: **x** and **y**, **width** and **height**.

When it comes to the property x of the object, we declare the value of the distance of our point from the beginning of the x-axis.

As for the property y of the object, we declare the value of the distance of our point from the beginning of the y-axis.

You can find the values that you need at the coordinates which are marked on our reference image.

The x and y values are located at the **upper left corner of each skyscraper**.

For **skyscWidth**, we subtract the value of **point x of the current** skyscraper from the value of **point x of the next skyscraper**.

For **skyscH**, we subtract from the **total height** of the canvas (height) the y value of the current skyscraper.

Like this:

```
var skysc1 = {
  x: 0,
  y: 321,
  skyscWidth: 158-0,
  skyscH: height-321
}

var skysc2 = {
  x: 158,
  y: 384,
  skyscWidth: 298-158,
  skyscH: height-384
}
```

Having created the 8 skyscraper objects, we are ready to go ahead with drawing them. First, we choose their color. Then we design the 8 rectangles using **rect (x, y, width, height);** and "invoking" the properties we defined in our objects like this **nameOfObject.nameOfProperty** .

```
//rect(x, y, width, height)
rect(skysc1.x,skysc1.y,skysc1.skyscWidth,skysc1.skyscH);
```

9) STEP 5:

We go back to the top of the code and define the variables we will need for the windows. We will need a variable for the length of each window side, ie the width and height of the square to be formed (**var windWidth = 20;**), one for the x value (**var windX**) and one for the y value (**var windY**). Upon the declaration of a new variable, it is not necessary to define their value straight away. At this point, the windWidth and windY variables will be left undefined because each window in our sketch has a different x and y point and it is more convenient to assign values later, separately for each skyscraper's windows.

Once we've created our variables, we scroll down again and choose the color of the buildings.

Before we begin designing the windows, we must initially declare the **x point** from which the windows for each skyscraper must start getting formed. This value is the **result of the sum** of each skyscraper's **point x value** plus **the distance between the edge of the wall of the skyscraper and the first row of windows**. We can find these values in our reference picture. Similarly, we have to declare the value of the variable of the **y point** from which the window's drawing should begin, but we'll explain how to do this later.

```
windX = skysc1.x+25;
```

Each building's first horizontal row of windows should be repeated across the skyscraper to give the illusion of multiple floors. To achieve this, we will use the "**for**" iteration structure, which should be written in the following way:

for (the 1st command indicates the **initial value** of the variable to be controlled by the repeat condition;**;** the 2nd command is the **condition whose validity is checked each time** so that the program knows if it will execute the code in the curly brackets (or, simply put, **the point until which the program will keep repeating the code in the curly brackets**); **;** the 3rd command is being executed at the end of each iteration and **changes the control variable**) **{**
Inside these brackets, we'll add the code that needs to be repeated.
}

In this case, the **control variable** is going to be the **windY** value of the first horizontal row of windows of each skyscraper, our **condition** is going to be whether the **windY value** is less than or equal to the total height of the canvas and as a 3rd command we'll **add 35 pixels to the windY** value, until the control variable becomes equal or as close as possible to the total height of the canvas which is 768px. This creates many triads of windows across the skyscraper's height every 35 pixels without having to design the windows one by one and find the exact y values of all windows.

Within the brackets of this “**for**” iteration structure, we will draw as many squares as should be in each row depending on how big each skyscraper is. We will design the windows using **rect()**; and with the help of the following **variables**: windX, windY and windWidth. We’ll be looking at our reference image to figure out the **distances between the windows** and **how many windows** we should draw **per row**. (This can also be achieved even faster with the use of nested for loops but it's a bit more complicated)

```
for (windY = skysc1.y+25; windY <= height; windY+=35) {  
  rect(windX,windY,windWidth,windWidth);  
  rect(windX+40,windY,windWidth,windWidth);  
  //+40 is the distance between the 1st and the 2nd rows of windows  
  rect(windX+80,windY,windWidth,windWidth);  
  //+80 is the distance between the 1st and the 3rd rows of windows  
}
```

10) STEP 6: As we’re approaching the end of this course, it's time to draw the hill. We will draw the hill using the **ellipse()** command to create a large oval shape. First, let's choose the color of the shape.

When it comes to the x value of this oval shape (AKA the center of our oval shape), we’re going to figure it out **by dividing** the **total width** of our canvas **by 2**. Thus, the center of our circle lies exactly in the middle of the horizontal axis (1024/2).

To give the impression of a hill, we only want to see the top half part of our oval shape. To do this, we’ll declare “y” as the y value of our canvas (the height of the canvas).

For this shape, we will set the values of the diameters in a way so that they overflow the canvas to create the feel of the hill (only the upper half of the oval shape is going to be visible.) We notice that only what lies within the canvas's dimensions is displayed in the browser. In order to know which values we will use, we’ll once again check our reference image.

11) STEP 7: At this stage, the sky is filled with stars at some point and it turns completely yellow, so we want to erase the previous stars once we get a reasonable amount of them and create new ones from the beginning, creating a nice effect. To do this, we have to find a way to calculate how many times the **function draw()** has been executed because this is what determines the number of the stars as well.

We will go to the top of the code again and set a new **variable** named **repeats** starting at **1** and increasing by one every time **draw()** is being executed. When draw() reaches **200 repetitions** (that is when 200 stars are created,) we will **erase the previous ones** and return the value of the **repeats to 1** to create the next round of 200 stars.

```
var repeats = 1;
```

We will use an if statement (selection structure). An if statement looks like this:

```
if (condition) {  
  function();  
}
```

The code in the curly brackets will only be executed if the condition in the parentheses is valid, that is, only if **draw()** has run 200 times and consequently the “repeats” variable is equal to 200.

In these brackets, we will put two commands. The first one will have to erase the stars and the second one to restart the repeats count. To erase the stars, we can simply redefine our **background** color, which will now fall over all of the previous stars and hide them. To restart the count, we will simply reinstate the value of **repeats = 1;**

Once we’re done with the if statement, we go back to the very end of our code and set that, each time the code reaches that point, the “repeats” variable will increase by one unit. This can be done in JavaScript using either **+= 1** or by adding **++** immediately after the name of our variable.

repeats + = 1;

CONGRATS! You’ve just learned some of the most basic concepts of programming and, all this while designing an animated drawing using code!