Reference: http://p5js.org/reference/

Introduction to p5js

P5.js is a JavaScript library that starts with the original goal of **Processing**, to make coding accessible for artists, designers, educators, and beginners.

Setting up the environment

Create a directory(You can name it whatever you want)

```
mkdir p5js
```

Navigate into the directory

```
cd p5js
```

Create a file named example1.js

```
touch example1.js
```

Create another file named index.html

```
touch index.html
```

open index.html with gedit and paste the following code in it

```
gedit index.html
```

```
<html>
	<head>
		<meta charset="UTF-8">
		<script language="javascript" type="text/javascript" src="p5.js"></script>
		<script language="javascript" type="text/javascript" src="example1.js"></script>
	</head>

	<body>
	</body>
</html>
```

Download a p5.js file from http://p5js.org/download/ and place it inside of the of the directory.
Now you have 3 files in total in that directory.

Introduction to programming with p5js

Note: When we were setting up the environment, we had 3 files, two that we created and a third that we downloaded. This code goes should be in the files we create that end with .js. Also index.html should be edited at this line

```
<script language="javascript" type="text/javascript" src="example1.js"></script>
```

to match the the .js file you are working with.
For you to view your changes, open index.html using your browser of choice.

Simplest program in p5js looks something like this

```
function setup() {
	//  this is where the statements go
}
function draw(){
	// this is where the statements go
}
```

The setup() → block **runs once**, and is typically used for **initialization.**

The draw() → block runs repeatedly, and is used for animation.

Creating and setting up the canvas

To create the canvas we use a function called createCanvas()
Example:

> createCanvas(700, 400)
> *description*
> *1st argument → 700 → this specifies the width of the canvas in pixels*
> *2nd argument → 400 → this specifies the height of the canvas in pixels*

This is how this would be in code

```
    function setup() {
        createCanvas(700, 400);
    }
```

Simple shapes without animation.

We will start by drawing simple shapes first giving a brief description of the functions that create the shapes.

1) Dot/Point

> point(25, 30);
>
> *Description*
> Draws a point, a coordinate in space at the dimension of one pixel. The first parameter(25) is the X coordinate for the point, the second parameter is the Y coordinate for the point.

2) Line

> line(15, 25, 70, 90);
>
> *Description*
> Draws a line (a direct path between two points) on the screen.
> 1St parameter → 15 → X coordinate of the first point.
> 2Nd parameter → 25 → Y coordinate of the first point.
> 3Rd parameter → 70 → X coordinate of the second point.
> 4Th parameter → 90 → Y coordinate of the second point.

3) Triangle

> triangle(30, 50, 18, 360, 81, 360);
>
> *Description*
> A triangle is made by 3 lines joined together by 3 points.
> 30, 50 → First point.
> 18, 360 → Second point.
> 81, 360 → Third point.

4) **Square/Rectangle**

> rect(81, 81, 63, 63);

Description
1st and 2nd parameters → 81, 81 → set the location of the upper-left corner of the rectangle or square.
3Rd parameter → 63 → this specifies the length of the rectangle/square.
4Th parameter → 63 → This specifies the width of the rectangle/square.

5) Circle/Oval

ellipse(252, 144, 72, 72);

Description
1st and 2nd parameters →252, 144 → Set the location of the center of the circle.
3Rd parameter → 72 → This specifies the horizontal length of the circle.
4Th parameter → 72 →This specifies the vertical length of the circle.

6) Trapezium/Rhombus

Find out

Functions that enhance shapes

fill()
Sets the color used to fill shapes.
noFill()
Disables filling geometry.
noStroke()
Disables drawing the stroke (outline of the shape).
stroke()
Sets the color of the stroke.

Note: We will draw those shapes on the canvas and Since our aim is to just draw the shapes (without animating them), then we will use the setup function.
Example code of how we draw the following shapes to the canvas

```
function setup() {
// this 2 lines of code create the canvas and sets it to a greyish color
createCanvas(720, 400);
background(80);

// This draws a line
line(15, 25, 70, 90);

// This draws a point
point(25, 30);

// This draws a triangle
triangle(30, 50, 18, 360, 81, 360);

// This draws a rectangle/square depending on the arguments
stroke(153);
fill(102, 12, 30);
rect(81, 81, 63, 63);
```

```
        // This draws weird four sided shapes like trapesiums/rhombus
        fill(100, 90, 30);
        quad(189, 18, 216, 18, 216, 360, 144, 360);

        // This draws circle or oval depending on the arguments you give it
        fill(255, 0, 0);
        ellipse(252, 144, 72, 72);

        // you can have another triangle
        fill(0, 255, 0);
        triangle(288, 18, 351, 360, 288, 360);

        // this draws a rectangle
        fill(0, 0, 255);
        rect(370, 60, 63, 300);

}
```

Comments

There are two types of comments possible in JavaScript

single line

```
// this is a single line comment
```

multi-line

```
/* This is a multiline comment */
```

Debugging in p5js

A bug is an error that might occur while you are programming.
There are several types of error in programming.

1) Syntax errors → error that occur if there is something wrong with your syntax. These errors are easily detected by the computer.

    Example: using small letters where you should be using capital letters.

2) Logical errors → these are errors that occur when your program does not do what you want it to do.
3) Runtime error →The runtime error is a specific type of error that can cause a program to terminate abnormally.

    example: dividing a number by zero is a nice example of a runtime error

4) Semantic errors → errors due to an improper use of program statements.
5) off-by-one error → Find this out

Inspect element

This is a very nice tool that that we will be very handy when you are debugging It comes with most browsers. You open it by right clicking on the page you are currently working on and clicking “inspect element” in the drop down menu that is displayed. Then select console on the top bar menu of the window that is opened.

console.log()

This is very handy function that one can use to display values in the console.

One simple shape animation.

frame rate.

For animation to occur in the computer it uses the concept of the frame rate. Frame rate is the frequency at which frames in a television picture, film, or video sequence are displayed.
In p5js animation is done in the draw function.
Example code of a simple animated circle that follows the mouse pointer.

```
function setup() {
  createCanvas(640, 480);
}

function draw() {
  if (mouseIsPressed) {
    fill(0);
  } else {
    fill(255);
  }
  //fill(0, 255, 0);
  ellipse(mouseX, mouseY, 80, 80);
}
```

Note: mouseIsPressed is an example of an in built/per-defined variable. An in built/pre-defined variable is a variable that has been written by some else for us to use and comes with the library you are using. MouseIsPressed is an in built variable that checks whether a variable is pressed or not. Other examples of in-built variables in the code are mouseX, mouseY.
ellipse is an example of an in-built function.

Variables in p5js

declaration and initialization

```
var nameOfVariable = 10;
```

Note: var is a keyword in programming. Keywords in programming are reserved words that have a standard, predefined meanings. Example of keywords in javascript → var, function.
Var is the keyword that means that a variable is being declared.

Different scope of variables

Global scope → A variable declared outside a function, becomes global and can be accessed in all the functions in the program.
Local scope → Variables declared within a function, are local to the function and can only be accessed within that function

example of using variables

Data types

There are several data types in programming.

1) Integers → whole numbers
2) Floats → numbers with decimal points
3) Strings → alpha-numeric characters in double or single qoutes

4) Boolean → Values with just two possible values usually true/false (1/0)

Operators

1) mathematical operators

+, -, *, /

2) comparison operators

> → Greater than

< → Less than

>= → Greater than or equal to

<= → Less than or equal to

== → Equal to

!= → Not equal to

3) boolean/logical operators

operate on boolean values

&& → This is the logical **AND**

The result of and is true only when one of the operands is true

Putting these in a simple table

	&&		**Result**
0	&&	0	0
0	&&	1	0
1	&&	0	0
1	&&	1	1

|| → This is the logical **OR**

The result of and is false only when both the operands are false

Putting these in a simple table

	||		**Result**
0	||	0	0
0	||	1	1
1	||	0	1
1	||	1	1

! →This is the logical **NOT**

This works with one operand. It negates the operand it is working on.

!	1	0
!	0	1

4) concatenation/join operators
This operator works on strings. It joins two strings together

5) Modulus operator.
This → %
is the modulus operator.
The modulus gives us the remainder of two numbers.

Example

```
var reminder = 2 % 3;
```

the remainder is 2

if conditions

General syntax

```
if (condition/boolean){
	// statements
}
```

Loading a background image.

The function that is responsible for making an image the background is the background().

```
background(bg);
```

Description

bg → This is a variable that contains a loaded image for the background.

To make an image a background start by creating a global variable to hold the image, then load the image, then make it the background.

Example in code

```
// Have a global variable to hold the background image
var bg;

function setup(){
	createCanvas(700, 400);

	// Load the image
	bg = loadImage("backgroundImage.png .png");
}

function draw(){
	clear();

	// Make the image the background
	background(bg);
}
```

Example code of how we can utilize the variable and if conditions with a background image
The task will be to try and keep a freely moving circle inside the canvas by making it bounce in all the four edges of the canvas.

```
// Declare two global variables for the x and y position of the circle
var x = 10;
var y = 10;

// Declare another two global variables to keep the x and y speed of the circle
var speedInY = 1;
var speedInX = 1;

function setup(){
        // These 2 lines of code create the canvas and load the background image
        createCanvas(720, 400);
        bg = loadImage("backgroundImage.png");
}

function draw(){
        //  These 2 lines clear the canvas after every frame and reload the background image
        clear();
        background(bg);

        // These 2 lines draw a circle and fill it with a greenish color
        fill(80, 255, 80);
        ellipse(x, y, 20, 20);

        // Update x  and y position of the circle such that it moves depending on the speedInX
        // and speedInY
        x = x + speedInX;
        y = y + speedInY;


        /* Check the Y position of the circle in relation to the top and bottom edges of the
           canvas.
            If the ball is at the top or bottom edge of the canvas, make it bounce back.
        */
        if (y > height - 10 || y < 0 + 10 ){
                speedInY = speedInY * -1;
        }

        /*TODO: Assignment
                Make the ball a bounce in the right and left edge of the canvas.
        */
}
```

Loading a sprite image

A sprite is just a normal image. The best function to display an image (which is not a background image) is the image() function.

```
image(sprite, 100, 120, 60, 60);
```

Description

sprite → This is a variable that contains a loaded image for the sprite.
100, 120 → These two arguments are the X and Y position (top-left corner) of the image on the canvas.
60 → This is the length of the image.
60 → This is the height of the image.

Example in code

```
// Have a global variable for the sprite image
var sprite1;

function setup(){
	createCanvas(700, 400);

	// Load the sprite image
	sprite1 = loadImage("sprite.png");
}

function draw(){
	clear();
	background(80);

	// display the sprite image on the canvas
	image(sprite1, 50, 50, 20, 20);
}
```

Controlling a spite using a keyboard

We will use keyIsDown() function for this.
The function used to detect if the player has pressed a certain key or not.

```
keyIsDown(LEFT_ARROW)
```

Description

keyIsDown →
The function itself is in the form of a question “is a certain key pressed?”. The answer can only be yes or no (a boolean value).

LEFT_ARROW →
This is the key that is the computer is checking whether or not its pressed.
LEFT_ARROW is an inbuilt variable that represents the left arrow on the keyboard.
There are other in built variables like; RIGHT_ARROW, UP_ARROW, DOWN_ARROW etc.

Example in code

```
// Declare two global variables to store the x and y positions of the sprite
var spriteXpos = 300;
var spriteYpos = 200;

// Have a global variable for the sprite image
var sprite1;
```

```
function setup(){
        createCanvas(700, 400);

        // Load the sprite image
        sprite1 = loadImage("sprite.png");
}

function draw(){
        clear();
        background(80);

        // Make the sprite move using left arrow
        if (keyIsDown(LEFT_ARROW)){
                spriteXpos -= 5;
        }
        // Display the image on the canvas
        image(sprite1, spriteXpos, spriteYpos, 20, 20);


        /*
        TODO:  Can you figure out how to make it move with
                right arrow
                up arrow
                down arrow
                        ??
        */
}
```

Note: This → spriteXpos -= 5; → is the shorthand for spriteXpos = spriteXpos – 5;

Looping/repeating

There are several types of loops in JavaScript but we first look at the for loop for now.

For loops

general syntax

```
for (initialization; condition; increment) {
   // statements to the repeated
}
```

- initialization → Set the start of the loop and it is executed before the code block starts.
- Condition → Defines the condition for running the loop. If the condition becomes false the loop stops.
- Increment → Is executed each time after the loop block has been executed and it purpose is to increment the initialized variable.

Example

```
for (var i = 0; i < 10; i++ ){
        // some statements
}
```

Example in code:
You can draw five circles like below.

```
function setup(){
	createCanvas(500, 300);
}
function draw(){
	background(90);

	// five circles
	ellipse(50, 100, 50, 50);
	ellipse(100, 100, 50, 50);
	ellipse(150, 100, 50, 50);
	ellipse(200, 100, 50, 50);
	ellipse(250, 100, 50, 50);

}
```

That is easy. What if you wanted to draw 100 circles. Well its no longer easy, right?
This is a nice place to utilize a for loop like below. Instead of drawing one circle after the other like above, you can use a for loop to draw as many circles as you might want.

```
function setup(){
	createCanvas(800, 300);
}
function draw(){
	background(90);

	// draw many circles.
	for (var i = 10; i<770; i += 10){
		ellipse(i, 100, 10, 10);
	}

}
```

Randomness

Some times you might want randomness in your program.
The function for this is random(). The function gives us a random number between a specified range.

```
random(min, max);
```

Description
min → The minimum value in the range of values.
max → The maximum number in the range of values.

Example in code:

```
function setup(){
	createCanvas(800, 300);
}
```

```
function draw(){
	background(90);
	// draw many circles.
	for (var i = 10; i<770; i += 10){
		// make the x - coordinate of the circle random
		var randomXpos = random(20, 50);
		ellipse(i, randomXpos, 10, 10);
	}

}
```

Note: The variable randomXpos is local to the for loop.

Detecting colors

We use the get() function to detect the color of a particular pixel.

```
get(20, 20)
```

Description
20, 20 → coordinates of a pixel.

Arrays

An array is a list of values/data of the same type.
Example:
A list of integers becomes a an array of integers.
Arrays are identified by square brackets.

General syntax
[value1, value2, value3, value4]

Example

```
[1, 2, 3, 4];
```

Storing an array in a variable

It is in most cases advisable to store an array in a variable. Like so,

```
var arrayOfIntegers = [1, 2, 3, 4];
```

Storing an empty array in a variable

An empty array is just an array without any value in it.

```
Var emptyArray = [];
```

Comparing two arrays

You might have two arrays and you want to know if they contain the same values.
Example:

```
var array1 = [1, 2, 3, 4];
var arrray2 = [1, 2, 3, 4];
```

There are many ways to do this but in my opinion the simplest is the method below.

1) Convert the first array into a string like so.

```
array1.join()
```

2) Convert the second array into a string also like so.

```
array2.join()
```

3) Then compare them like so

```
array1.join() === array2.join()
```

One shape bouncing off another shape

Example code

```
// X and Y positions of the ball
var ballXpos = 100;
var ballYpos = 100;
// X and Y speeds of the ball
var ballXspeed = 5;
var ballYspeed = 5;

// X and Y positions of the ball
var brickXpos = 500;
var brickYpos = 50;

// Y speed of the ball
var brickYspeed = 0.5;

function setup(){
        createCanvas(800, 300);
}

function draw(){
        background(90);

        // draw rectangle
        fill(255, 255, 0);
        rect(brickXpos, brickYpos, 20, 100);

        /*get the color of the pixel at the middle of the brick/rectangle
                20/2 is half width of the rectangle
                100/5 is half height of the rectangle
        */
        var colorOfBrick = get(brickXpos + 20/2, brickYpos + 100/ 2);

        // draw the circle
        fill(255, 10, 100);
        ellipse(ballXpos, ballYpos, 50, 50);

        /* get color of the circle at the left and right edges of the circle
          Note: The diameter of the circle is 50 therefore its radius is 25
                            We can therefore know the color of the pixel at the edge of a circle
        */
        var colorCicleEdge1 = get(ballXspeed > 0? (ballXpos + 28): (ballXpos - 28) , ballYpos);

        // Get the color of the circle at the top and botton edges of the circle
        var  colorCicleEdge2 = get(ballYspeed > 0? (ballYpos + 28): (ballYpos - 28) , ballYpos);


        /* This compares if color at the side edge of the circle is the same as the color of the brick
                if they are the same bounce of the brick
        */
        if (colorOfBrick.join() === colorCicleEdge1.join()){
                ballXspeed *= -1;
        }


        /* This compares if color at the top and or bottom edges of the circle is the same as the color of
the brick
                if they are the same bounce of the brick
        */
        if (colorOfBrick.join() === colorCicleEdge2.join()){
                ballXspeed *= -1;
        }

        // the ball bounces off the top and bottom edge of the canvas
        if (ballYpos > height - 26 || ballYpos < 0 + 26){
                        ballYspeed *= -1;
        }

        // the ball bounces off the left and right edge of the canvas
        if (ballXpos > width - 26 || ballXpos < 0 + 26){
                        ballXspeed *= -1;
        }

        // the brick bounces off the  top and bottom edge of the canvas
        if (brickYpos > height - 100 || brickYpos < 0){
                        brickYspeed *= -1;
        }

        // Makes the ball and the brick move
        ballXpos += ballXspeed;
        ballYpos += ballYspeed;
        brickYpos += brickYspeed;
}
```