Betanode Workshop

Jan 12, 2021

Index

CHAPTER 1

Introduction

1.1 What is p5.js?

p5.js is a JavaScript library. In a nutshell, it lets us do a lot of cool stuff without writing much code. That's all you need to know (for now!)

1.2 Hello World!

Hello world is p5js is slightly different than in most other languages. We draw a line from the Origin (0, 0) to a point (400, 400). Type the following into your sketch.js file:

```
function setup() {
    createCanvas(1000, 500);
}

function draw() {
    // Draw a line
    line(0, 0, 200, 200);
}
```

In this snippet, we create 2 functions (more on that later): *setup* and *draw*. When our script is run, the function setup is called and the code inside is executed. This means the *createCanvas* line is called, which creates a canvas of width 1000 pixels and height 500 pixels.

Then, the *draw* function is called. Inside the draw function, the *line* function is called which draws a line from the origin (0, 0) to a point (200, 200).

1.3 Variables

Variables are containers that store data. This containers can store any type of data, be it a name, a mobile number, an Aadhar number, an email address, and so on.

The basic syntax to create a variable is:

```
let my_varaible_name = my_variable_value;
```

For example,

```
let my_number = 1234567890;
let my_email_id = 'ishanmanchanda70@gmail.com';
```

In Javascript, you can also create a variable by saying *var* instead of *let*.

1.4 'main' and 'setup'

main and *setup* are special functions.

CHAPTER 2

Control Flow

2.1 Conditionals

Conditionals allow programmers to perform certain tasks when a particular condition is met. They are a fundamental part of most programming languages. Conditionals are a usually implemented through an if…else… structure. They control behavior and determine whether or not some pieces of code should run.

For example, in JavaScript,

```
if (hour < 12) {
    greeting = "Good Morning"
} else if (hour < 17) {
    greeting = "Good Afternoon"
} else {
    greeting = "Good Evening"
}
```

2.2 Loops

Loops tell the computer to perform a certain task repeatedly, a certain number of times or until a condition is met. Very often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

There are two different kinds of loops in JavaScript,

for - loops through a block of code a specified number of times

```
for (let v = start; v <= end; v += increment) {
        // code that is run multiple times.
}
```

To control the behaviour of the for loop we use an increment variable, in this case it's 'v'. **while - loops through a block of code while a specified condition is true.**

```
while (v <= end_value) {
    // code that is executed until the above condition in met
    // Common Mistake: If you don't change the value of v in here, you will have an␣
↪infinite loop!
    // Make sure you avoid doing that since it can be difficult to find the issue␣
↪(debug) as a beginner.
}
```

The *do…while…* loop is variant of the while loop, it contains a block of code that is executed once, then a while loop is executed.,

```
do {
   // code that is executed once
} while (v <= end_value) {
   // code that is executed multiple times.
}
```

There are still other variation of loops such as the 'for…in…' loop, but they all follow the same basic principles.

CHAPTER 3

Some Examples

3.1 Example 1: Bouncing Ball

3.2 Example 2: [Your idea here!]

CHAPTER 4

Functions

4.1 What is a function?

- A function is a subprogram designed to perform a particular task.
- Functions are executed when they are called. This is known as invoking a function.
- Values can be passed into functions and used within the function. This is done through arguments that the function may accept
- Functions can return a value. In JavaScript, if no return value is specified, the function will return *undefined*.
- Javascript contains some built-in functions, external libraries like p5 expand the set of these functions.

4.2 Defining a Function

To define a function the basic syntactical structure is used,

```
function name(parameters) {
    // code in the function
}
```

4.3 Calling a Function

Calling/Invoking a function asking it to run at the point at which you have called it. Think of it like excecuting a keyboard shortcut, the function to make a new window is defined, and when the computer gets 'ctrl+N' it calls (runs) the new window function.

Syntax,

```
functionName(arguments);
```

If your function was defined without parameters,

```
functionName();
```

CHAPTER 5

Data Structures

5.1 Arrays

Arrays are a type of data structure in javascript. They are used to store multiple values into a single variable. Eg,

```
var array_name = [item1, item2, ...];
```

To access items in an array,

```
let item1_value = array_name[0];
// Here we put the index of the value we want to access in square brackets.
```

Using the same syntax to access items in an array, we are also able to change items.

```
array_name[0] = 'some value'
```

5.2 Objects

In the above context objects are a more general form of an array, the major difference is, instead of using index values to get items, objects use the following syntax,

```
 // defining the object
 var person = {firstName: "John", lastName: "Doe"; age:34};

// accessing items
let fullName = person.firstName + " " + person.lastName
```

5.3 Vectors

p5 has a builtin function that allows the definition of vectors. (An entity that has both, direction and magnitude, which can be accessed with the mag() and heading() methods respectively.) These are especially useful when one is simulating motion. To create a vector,

```
createVector([x], [y], [z])
```

Since vectors represent groupings of values, we cannot simply use traditional addition/multiplication/etc. Instead, we'll need to do some "vector" math, which is made easy by the methods inside the p5.Vector class.

The p5.vector has many useful methods that we will use frequently. Here are some listed:

- copy()
- add() // to perform vector addition
- sub()
- mult() // multiply the vector by a scalar
- div()
- mag() // calculates magnitude of the vector
- magSq() // xx + yy + zz
- dot() // dot product
- cross // cross product
- dist()
- normalize()
- heading()
- rotate()
- angleBetween()
- equals()

CHAPTER 6

Classes and Objects

6.1 Object Oriented Programming

OOP is a programming paradigm to make complicated code easier to write and understand, this will be a very brief description of OOP.

OOP is a programming paradigm based on the concept of objects, which may contain contain data(characteristics) and code in the form of procedures(behaviours) known as methods.

OOP contains two major parts, * classes * objects

A **class**, in object-oriented programing, is a code template for creating objects. The template contains some basic characteristics/intial values and implementations of behaviour of an object.

An **object** refers to a particular instance of a class, where the oject can be a combination of Variables, functions(methods) and data structures. Really an object is just a data type with Polymorphism and Inheritance (Wikipedia them)

For all intensive purposes, we can understand **methods** as functions that perform certain instructions on an object, the way to allow us the modify the behaviour of the object.

This will become the Wikipedia article on OOP soon, so this is all the theory.

Imagine creating a video game character in code

We could keep making variable to represent each characteristic, but that will get very cumbersome and unintuitive once there is more than one character.

So, we use a class to create a structure for the definition of the character.

```
class Character {
            constructor(name, level, max_health, current_health, max_speed, current_
↪speed) {
                    this.name = name;
                    this.level = level;
                    this.max_health = max_health;
                    this.current_health = current_health;
```

(continues on next page)

(continued from previous page)

```
                    this.max_speed = max_speed;  // We could also use a ps.Vector␣
↪object to represent velocity here
                    this.current_speed = current_speed;
            }
    }
```

A constructor function initializes all the required characteristics of the class.

Continuing with our example,

```
class Character {
            constructor(name, level, max_health, current_health, max_speed, current_
↪speed) {
                    this.name = name;
                    this.level = level;
                    this.max_health = max_health;
                    this.current_health = current_health;
                    this.max_speed = max_speed;  // We could also use a ps.Vector␣
↪object to represent velocity here
                    this.current_speed = current_speed;
            }

        sprint() {
            this.current_speed += 25
        }

        take_damage() {
            this.current_health -= 25
        }
    }
```

CHAPTER 7

More Examples

7.1 Example 1: Sine Wave

*y = A * sin(kx - phi)*, where:

A = Amplitude, k = Phase Difference between adjacent particles, x = Position of the particle, phi = Phase

7.2 Example 2: Brownian Motion

Brownian motion is random motion of particles. We will simulate random motion by giving particles an a random initial velocity. After that, each time draw is called, we will apply a random acceleration to the particle.

We will use pVectors to represent both:

```
velocity.add(acceleration);
```

7.3 Other Examples

- Patterns: *draw an image with a software tool which responds to the speed of the mouse.*
- Simulating the solar system.
- Making a simple pendulum.
- Colliding Spheres.
- Simulating free fall in different materials.
- **Snake**: If time permits, we will design (live!) the Snake game to demonstrate how to break a complex problem into easily-solvable components.