Os estudos efetuados aqui sobre P5.js são baseados principalmente nos vídeos sobre o tema do canal Coding Train.  
( <https://www.youtube.com/watch?v=HerCR8bw_GE&list=PLRqwX-V7Uu6Zy51Q-x9tMWIv9cueOFTFA&index=1> )

Iniciando:

P5.js é uma biblioteca de Java Script que nos permite manipular alguns elementos gráficos, é principalmente utilizada mostrar dados mas também pode ser utilizada para criação de jogos ou coisas do gênero.

A principal referencia sobre P5.js é o site https://p5js.org/ .

Para poder utilizar a biblioteca existem duas maneiras:

1. Baixar e utilizar na IDE desejada;

2. abrir o web editor de P5.js;

A segunda maneira costuma ser mais pratica pelo menos no qu tanje a aprendizagem, logo, será a maneira utilizada. Para tal basta acessar <https://editor.p5js.org/>, recomenda-se que tenha uma conta no site para poder salvar os códigos.

Uma coisa muito interessante do web editor é que ele organiza o código de maneira que fique mais legível, basta aperta shft+tab.

Trabalhando com P5.js:

Aqui cabe dizer, sendo o P5.js uma biblioteca de JavaScript, ele obedece a toda a sintaxe de javascript, desta maneira, ele obedece a todos os comandos e ao se programar usando essa biblioteca é importante seguir todos os paradigmas implícitos à esta linguagem como OOP e etc...

Formas e Desenhos:

Em P5.js se tratando de uma biblioteca, com ele existem várias funções que se destacam, as principais sendo relacionadas a capacidade de desenhar no canvas em html criado pelo editor formas, veja:

function setup() {

    createCanvas(400, 300);

  }

  function draw() {

    background(220, 0, 200);

    line(0, 50, 400, 300);

    rectMode(CENTER);

    rect(200, 150, 150, 150);

  }

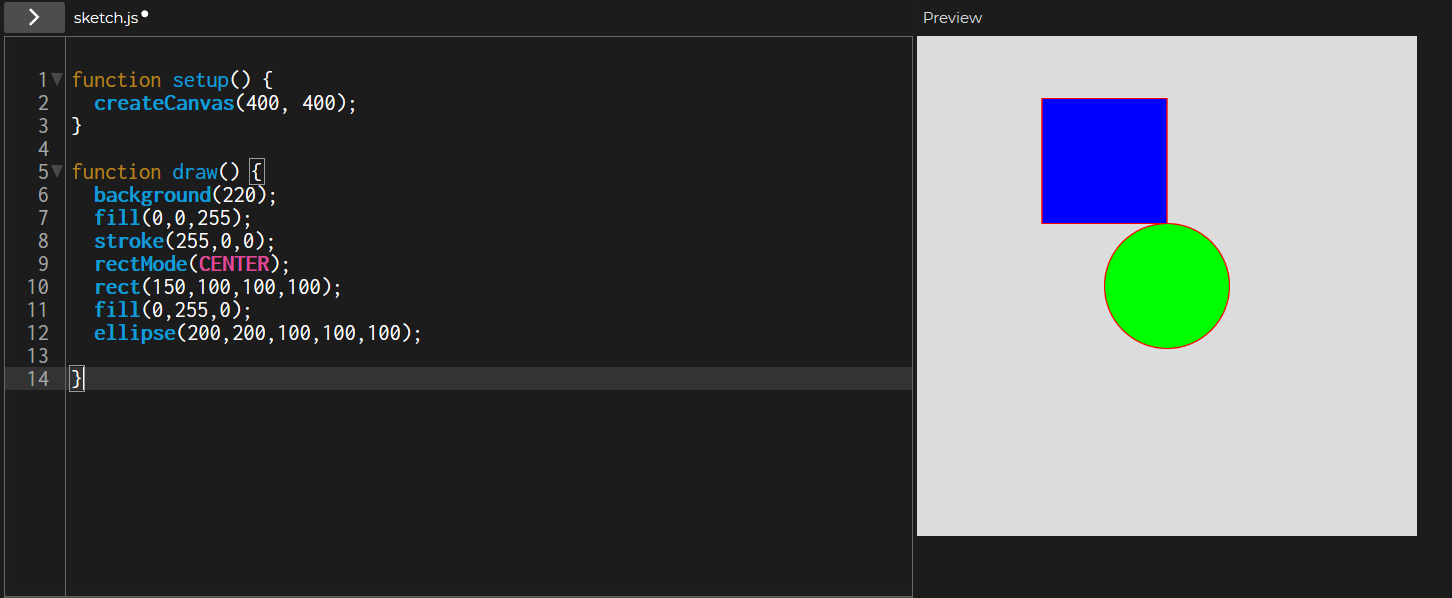
Aqui foram demonstrados apenas as formas de retângulo (rect) e a linha (line) porem existem algumas outras como o arco (arc) e a elipse (elipse), é encorajado que se procure as documentações pertinentes, em <https://p5js.org/reference/>, para entender melhor como funcionam.

Cores:

A principal maneira como definimos as cores em P5 é através do sistema RGB, em que os valores variam de 0 à 255, há maneiras de alterar a forma como esses valroes se comportam mas esta é a principal.

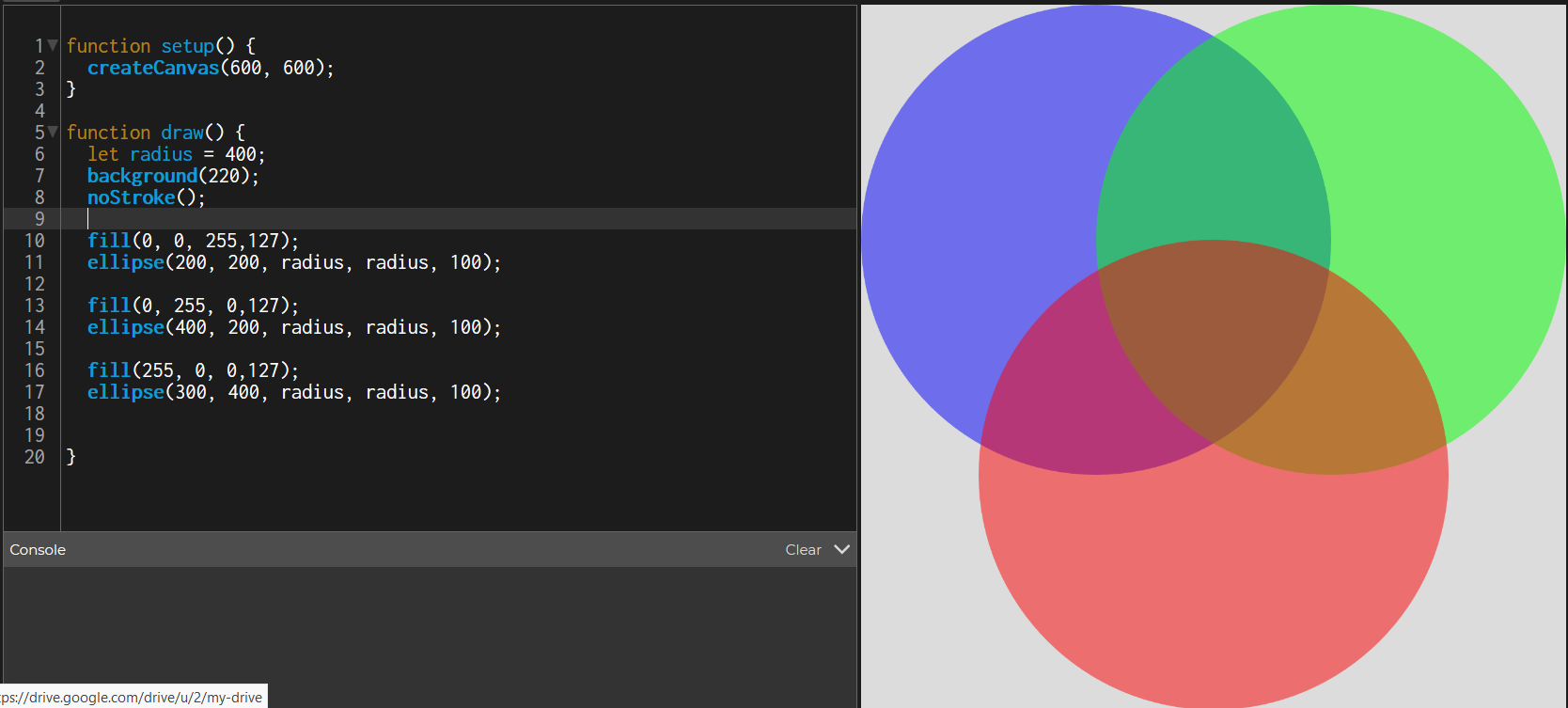
As principais funções envolvendo cores de uma forma é o fill() e o stroke().

O fill() lida com a parte interna do objeto em quanto o stroke lida com a parte externa (outLine) dele.



Ambos as funções quando utilizadas alteram os valores das cores dos objetos seguintes até que um novo comando seja escrito. Isso acontece pois o p5.js, assim como o Java Script em si, funciona como se fossem “camadas”, o que está em cima executa primeiro e altera oq está por baixo.

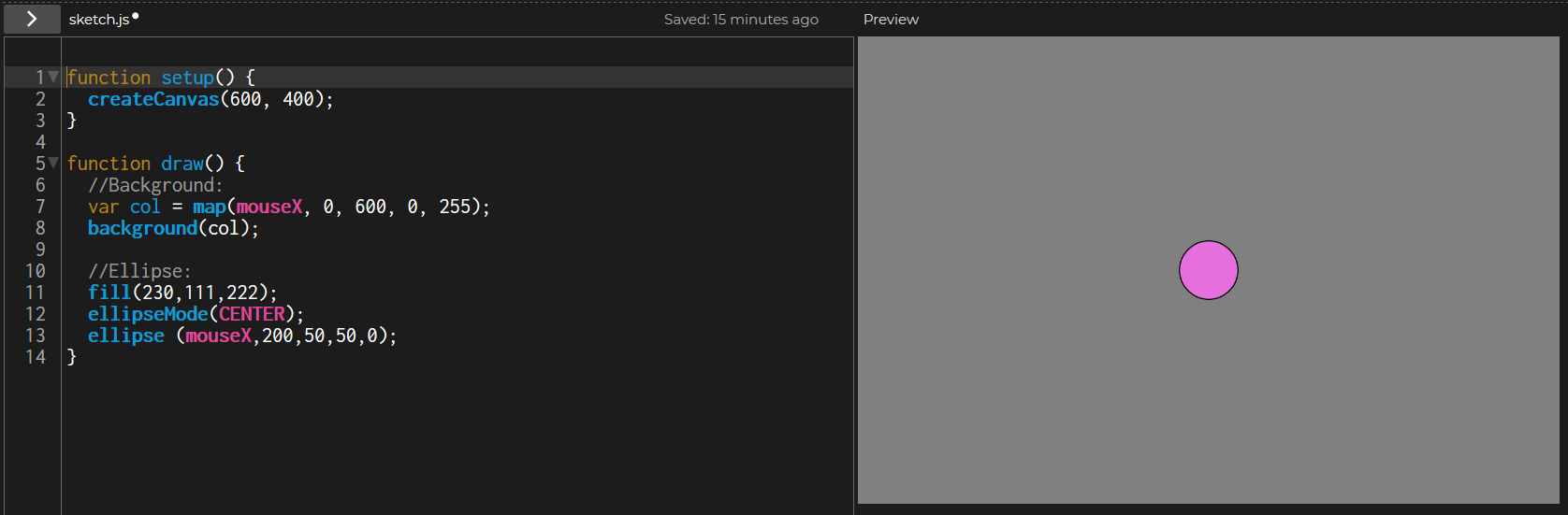
Ainda há mais um argumento que podemos passar para essas duas funções que define a opacidade do objeto. Esse quarto parâmetro é assumido como sendo 255 quando não utilizado, ou seja, quanto menor for o valor da opacidade mais transparente o elemento será. Veja:



Perceba também que o ‘background()’ está utilizando apenas um parâmetro para sua cor, quando isso acontece a biblioteca assume que está lidando com uma escala de cinza de 0 à 255 em que 0 é branco e 255 é preto.

Map():

A função ‘map()’ basicamente faz uma regra de 3 e converte um valor que está em um determinado range e transforma esse valor em outro range. O Exemplo a seguir demonstra bem isso, ele faz com que o background altere sua cor de branco para preto conforma a posição X do mouse na tela, veja:



O ‘map()’ possui 5 argumentos:

Map( variável que estou dando track,

Mínimo da range que estou dando track,

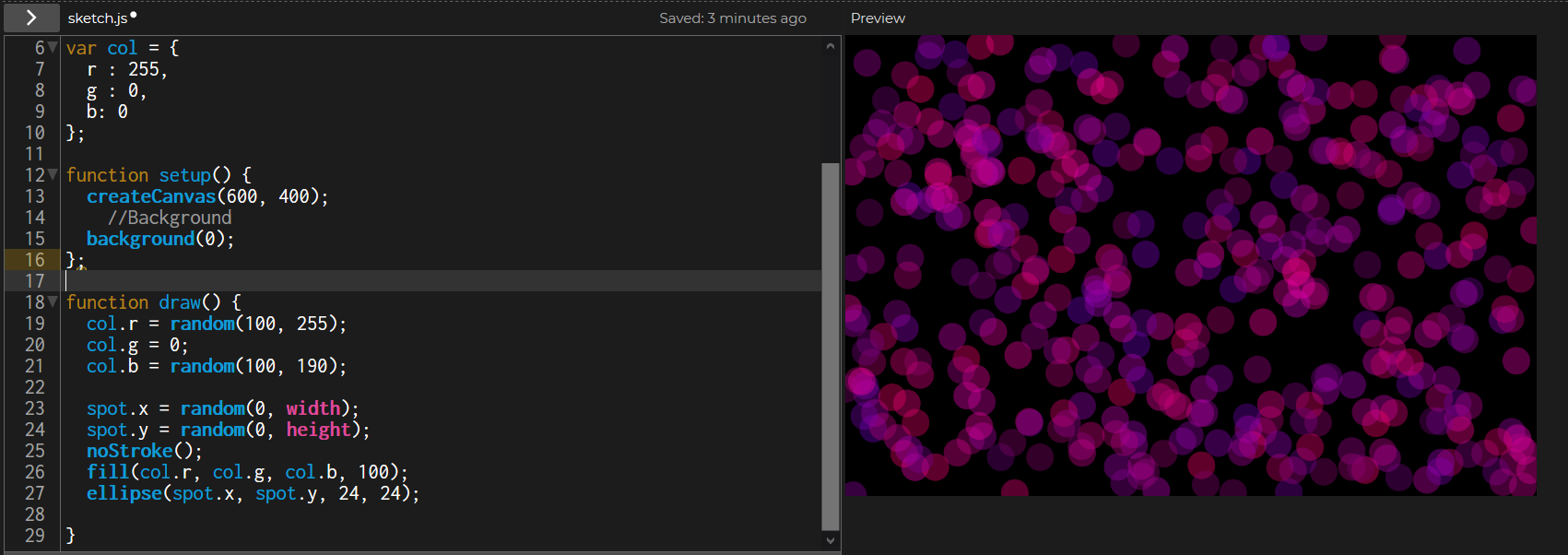
Maximo da range que estou dando track,

Mínimo da range pretendida,

Maximo da range pretendida)

Random():

Essa função pode ser usada somente dentro do draw() ou do setup(). Ela leva dois argumentos, o primeiro sendo o valor mínimo aleatório a ser gerado e o segundo sendo o valor máximo a ser gerado. Veja no exemplo a seguir como ele funciona:



INTERAÇÕES:

Este capitulo lida com expressões booleanas, que retornam/dependem de valores True ou False.

If/else:

O IF/ELSE funciona da mesma maneira que em qualquer outra linguagem, se a expressão escrita após o IF o código que vem na identacao do IF é executado, caso seja falso, não é executado.

// Expressoes Booleanas: IF/ELSE

function setup() {

  createCanvas(600, 400);

};

function draw() {

  background(0);

  stroke(255);

  strokeWeight(4);

  noFill();

  //Inicio do bloco IF

  if (mouseX > 300) {

    fill(200, 0, 200);

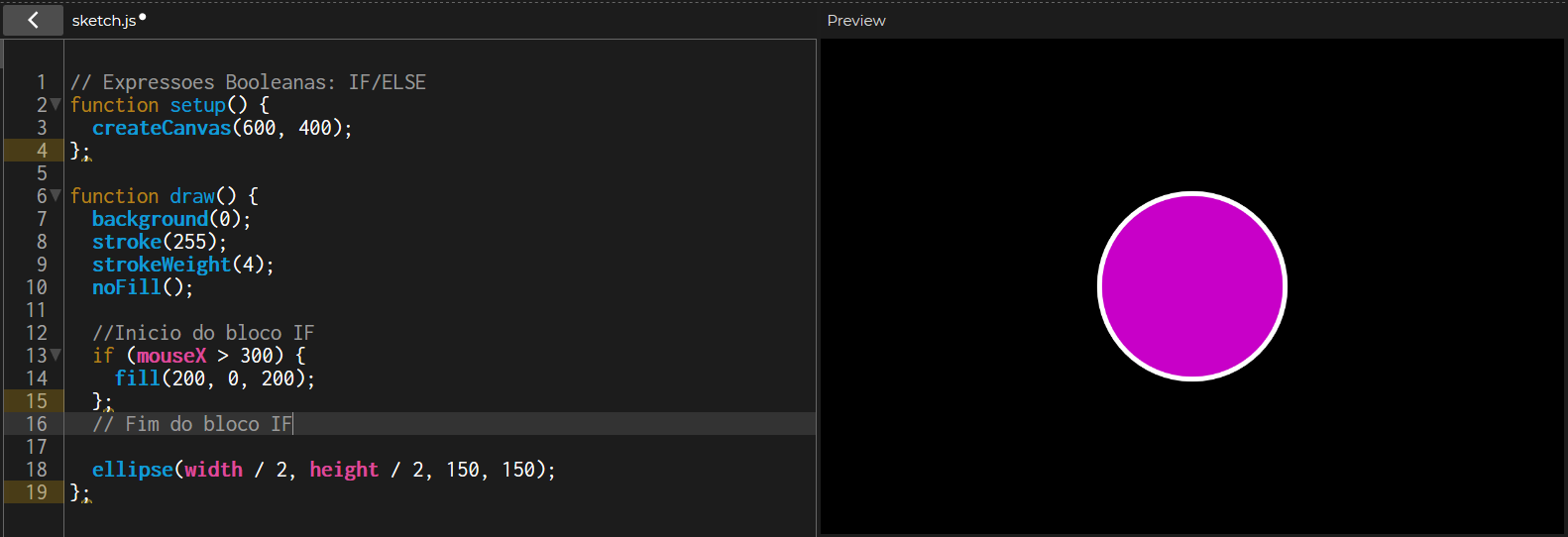
  };

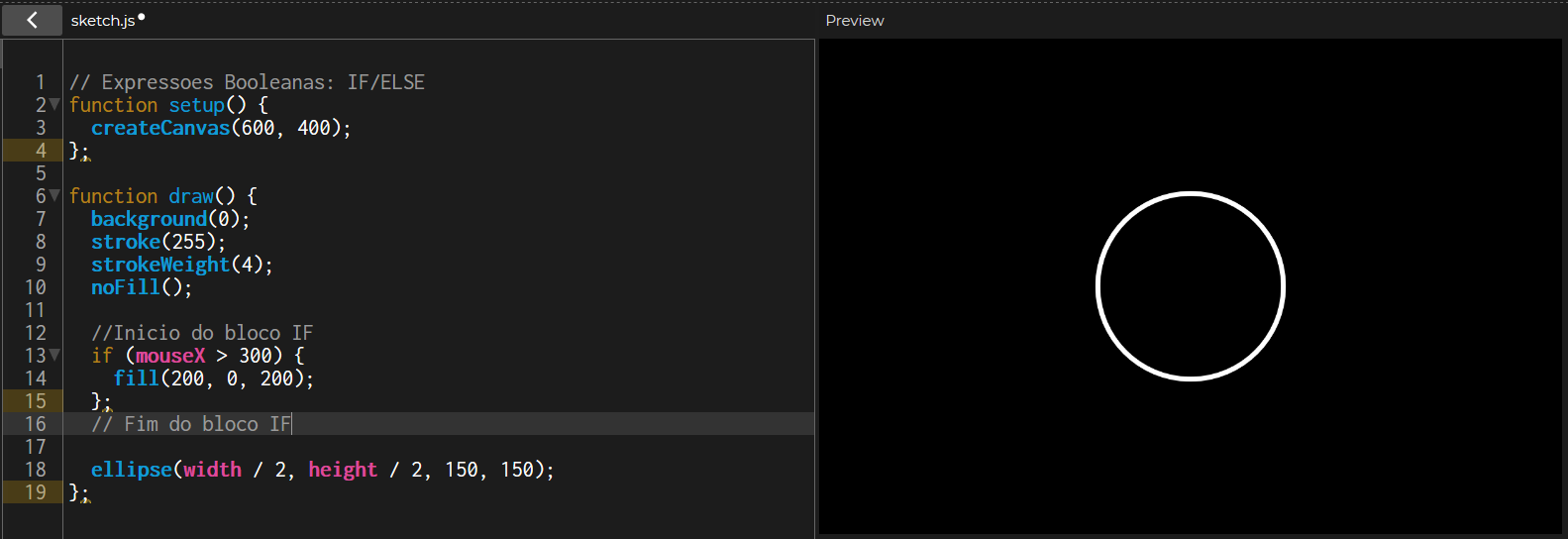
  // Fim do bloco IF

  ellipse(width / 2, height / 2, 150, 150);

};

Neste exemplo, fica clara a sintaxe a ser utilizada para blocos IF/ELSE, o programa em questão preenche o circulo criado no centro do canvas com uma cor roxa caso o mouse esteja no lado direito do canvas e caso esteja no lado direito ele não tem preenchmento.





Aqui está um exemplo de um código que cria um circulo em um x e y aleatórios no cnavas e faz esse circulo quicar nos cantos:

//Bola que quica

var x = 0;

var y = 0;

var speedx = 10;

var speedy = 10;

function setup() {

  createCanvas(600, 400);

  x = random(0, width / 2);

  y = random(0, height / 2);

  //background(0);

}

function draw() {

  background(0);

  ellipseMode(CENTER);

  noFill();

  stroke(255);

  strokeWeight(4);

  ellipse(x, y, 50, 50);

  //Quicada em X

  if (x < 0) {

    speedx = -speedx;

  } else if (x > width) {

    speedx = -speedx;

  }

  // Quicada em Y

  if (y < 0) {

    speedy = -speedy;

  } else if (y > height) {

    speedy = -speedy;

  }

  y = y + speedy;

  x = x + speedx;

}

Ainda é possível adicionar dois adicionais aos if/else`s: AND e OR. Como os nome sugerem, quando colocados nas condições do if funcionam como operadores lógicos do tipo E ou do tipo OU.  
-sintaxe do AND: &&;  
-sintaxe do OR: ||;

Veja:

//AND e OR

function setup() {

  createCanvas(400, 400);

}

function draw() {

  background(220);

  // OR

  if (width < mouseX || height < mouseY) {

    fill(255, 0, 0)

  }

  // AND

  // if (200 < mouseX && 200 < mouseY) {

  //   fill(0, 255, 0)

  // }

  else {

    fill(0, 0, 255)

  }

  ellipse(width / 2, height / 2, 150, 150)

}

Valores Booleanos:

Valores booleanos são valores True ou False, algumas funções tem a característica de retornar um desses dois valores, são uteis qunado o código precisa checar se uma informação é verdadeira ou falsa (apenas dois estados). Veja no exemplo a seguir a utilização do ‘mouseIsPressed’:

/// Interruptor de luz

var lightSwitch = false

function setup() {

  createCanvas(400, 400);

}

function draw() {

  if (lightSwitch) {

    background(255);

  } else {

    background(0);

  }

  rectMode(CENTER);

  noFill();

  strokeWeight(4);

  stroke(127);

    rect(width / 2, height / 2, 100, 100)

}

function mousePressed() {

  if (mouseX < 250 && mouseX > 150 && mouseY < 250 && mouseY > 150) {

     lightSwitch = !lightSwitch

  }

}

Veja que na função ‘MousePressed()’ utilizamos o operador ‘not (!)’, ao pressionar a tecla do mouse na área delimitada pelo ‘if’ (para simular um botão) a variável ‘lightSwitch’ recebe seu valor contrario (“interruptor recebe o valor inverso de interruptor”), sendo assim, se ele esta true ele muda para false e vice-versa; isto é uma alternativa mais elegante para o código do tipo

function mousePressed() {

  if (lightSwitch == true) {

    lightSwitch = false;

  } else {

    lightSwitch = true

  }

Que além de levar apenas uma linha para ser escrito é mais legível.

Loops:

Existem dois tipos de loops utilizados em JS e consequentemente em P5.js: for e while.

O while realiza o código identado até que certa condição seja atingida; o for cria uma variável própria que é manipulada pelo loop e então realiza o código identado um numero N de vezes baseado nesta vriavel, veja o exemplo:

//Loops for e while

function setup() {

    createCanvas(600, 400);

  }

  function draw() {

    background(0);

    var x = 0;

    //desenha bolinhas de golfe na horizontal

    while (x <= 600) {

      fill(255);

      ellipse(x, height / 2, 50, 50);

      x += 50;

    }

    //desenha bolinhas de golfe na vertical

    for (var y = 0; y <= height; y += 50) {

      fill(255, 255, 0);

      ellipse(width / 2, y, 50, 50);

      y += 50;

    }

  }

PS: Existe ainda uma outra forma d eusar um loop ‘for’ quando você quer iterar algum código para todos os itens de um array, chamaremos essa forma de ‘for...of’, veja:

//For ... of

  for (let item of obj) {

    item.display();

    item.move();

  };

Ele diz: “para cada ‘item’ de ‘obj’ execute: item.display() e item.move()”

Nested Loops:

São loops dentro de loops que fazem o seguinte: para tal variável de tanto a tanto faça 🡪 para tal variável de tanto a tanto faça tal coisa; Veja no exemplo a seguir como um nested loop de for desenha uma malha X:Y de bolinhas:

//Desenha uma malha de Bolinhas

var offset = 0;

function setup() {

  createCanvas(600, 400);

}

function draw() {

  background(0);

  noStroke();

  // desenha uma malha

  for (var x = 0; x <= width; x += 50) {

    for (var y = 0; y <= height; y += 50) {

      fill(random(127), 0, random(127));

      ellipse(x + offset, y, 25, 25);

    }

  }

}

FUNÇÕES:

Funções são blocos modulares que executam um determinado código.

Definir e chamar são dois termos associados a funções que respectivamente querem dizer “criar” e “usar” a função, no código a seguir vemos um exemplo tanto da sintaxe para sua criação como do uso de uma função.

//Bola que quica utilizando funcoes

var ball = {

    x: 50,

    y: 50,

    xspeed: 3,

    yspeed: -4

  }

  function setup() {

    createCanvas(400, 400);

  }

  function draw() {

    background(0);

    move(ball);

    bounce(ball);

    display(ball);

  }

  function move(ball) {

    ball.x += ball.xspeed;

    ball.y += ball.yspeed;

  }

  function bounce(ball) {

    if (ball.x > width || ball.x < 0) {

      ball.xspeed = -ball.xspeed;

    }

    if (ball.y > height || ball.y < 0) {

      ball.yspeed = -ball.yspeed;

    }

  }

  function display(ball) {

    stroke(255);

    strokeWeight(4);

    noFill();

    ellipse(ball.x, ball.y, 24, 24);

  }

Perceba que quando separamos o código de maneira modular, isto é, em funções, ele além de ficar mais legível fica mais fácil de trabalhar; basta declarar as funções e utiliza-las de maneira coerente.

OBS: aqui o autor dos vídeos faz menção ao livro Get Programming with P5.js que é um ótimo recurso para se estudar P5.

Veja mais um exemplo:

// Desenha um Pirulito

function setup() {

    createCanvas(400, 400);

  }

  function draw() {

    background(0);

    lollipop(width/2, height/2, 100);

    lollipop(50,50,50);

  }

  function lollipop(x, y, headSize) {

    fill(0, 200, 255);

    rect(x - 10, y , 20, 150);

    fill(250, 0, 200);

    ellipse(x, y, headSize, headSize);

  }

CLASSE:

Uma classe é uma maneira de encapsular valores e funções a um conjunto de objetos.

Funciona basicamente da mesma maneira que funções construtoras em JS mas funciona como um jeito melhor de organizar o código pois criando um objeto de uma determinada classe ele terá os atributos daquela classe.

//Classes:

function setup() {

    createCanvas(700, 500);

    bubbles = {}

    for (var i = 0; i < 50; i++) { // Cria-se 50 bolhas

      bubbles[i] = new Bubble;

    }

  }

  function draw() {

    background(0);

    for (var i = 0; i < 50; i++) { // excuta a funcao move() e show() para cada uma das 50 bolhas criadas

      bubbles[i].move();

      bubbles[i].show()

    }

  }

  class Bubble {

    constructor() { //Inicializa os parametros que o objeto pertencente a essa calsse terao

      var topSpeed = 10;

      var minSpeed = 0.1;

      this.x = random(width);

      this.y = random(height);

      this.xspeed = random(minSpeed, topSpeed);

      this.yspeed = random(minSpeed, topSpeed);

      this.size = random(10, 25);

      this.r = random(255);

      this.g = random(255);

      this.b = random(255);

        }

    move() { // funcao que a classe Bubble terá

      this.y += this.yspeed;

      this.x += this.xspeed;

      if (this.x > width || this.x < 0) {

        this.xspeed = -this.xspeed

      }

      if (this.y > height || this.y < 0) {

        this.yspeed = -this.yspeed

      }

    }

    show() {

      noStroke();

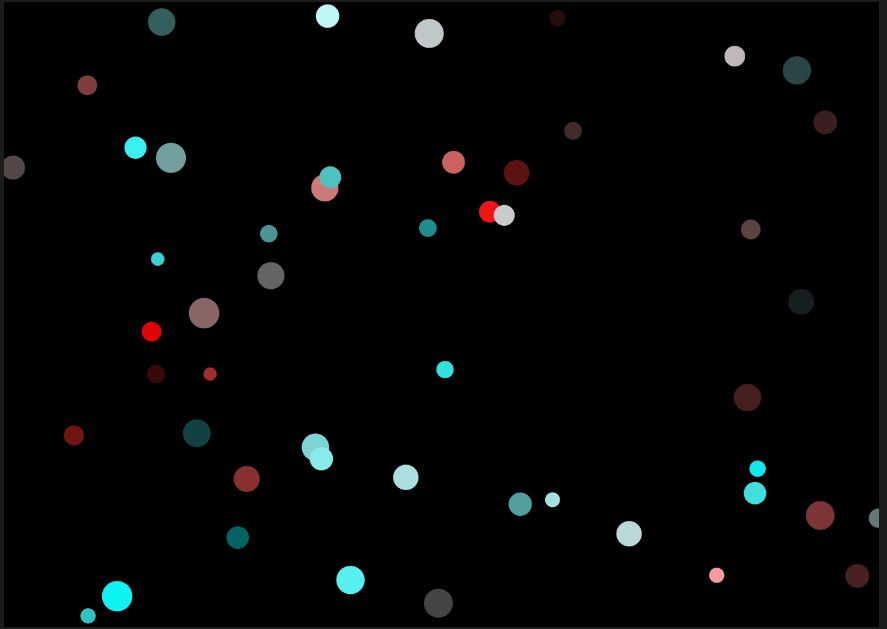
      fill(this.r, this.b, this.b);

      ellipse(this.x, this.y, this.size, this.size);

    }

  }

Resultado:



Perceba que neste exemplo não usamos nenhum argumento na criação das bolhas, porem caso desejássemos escolher o tamanho, por exemplo, basta passar seus argumentos para a função construtora, veja o recorte apenas da parte da classe para o codgio anterior:

//Classes: exemplo setando o parametro 'size' das bolhas

function setup() {

    createCanvas(700, 500);

    bubbles = {}

    for (var i = 0; i < 50; i++) {

      bubbles[i] = new Bubble(15); //O argumento passado aqui sera o argumento utilizado pelo constructor()

    }

  }

  .

  .

  .

  class Bubble {

    constructor(tempSize) {

      var topSpeed = 10;

      var minSpeed = 0.1;

      this.x = random(width);

      this.y = random(height);

      this.xspeed = random(minSpeed, topSpeed);

      this.yspeed = random(minSpeed, topSpeed);

      this.size = tempSize;

      this.r = random(255);

      this.g = random(255);

      this.b = random(255);

        }

  }