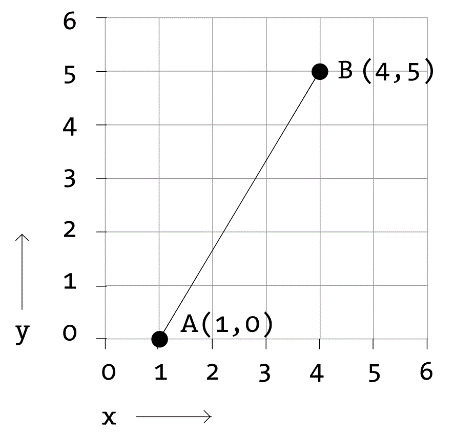
**План-конспект JS**

**Система координат і фігури**

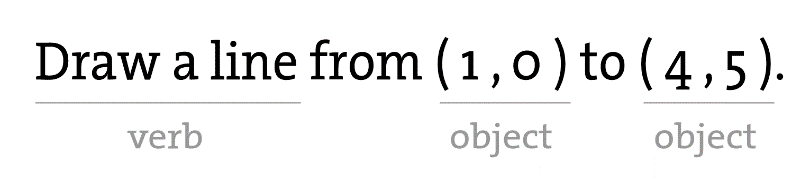
Перш ніж ми почнемо програмувати з p5, ми повинні спершу накреслити лінію. Найкоротша відстань між двома точками — це стара добра лінія, і ми починаємо з неї дві точки на міліметровому папері.



На малюнку вище показано лінію між точкою A (1,0) і точкою B (4,5). Якби ви хотіли наказати своєму другові намалювати ту саму лінію, ви крикнули б йому та сказали: «Проведи, будь ласка, лінію від точки один-нуль до крапки чотири-п’ять». Ну, а зараз уявіть, що ваш друг — комп’ютер, і ви хочете наказати цьому цифровому приятелю відобразити той самий рядок на екрані. Застосовується та сама команда. Тут інструкція буде виглядати так:

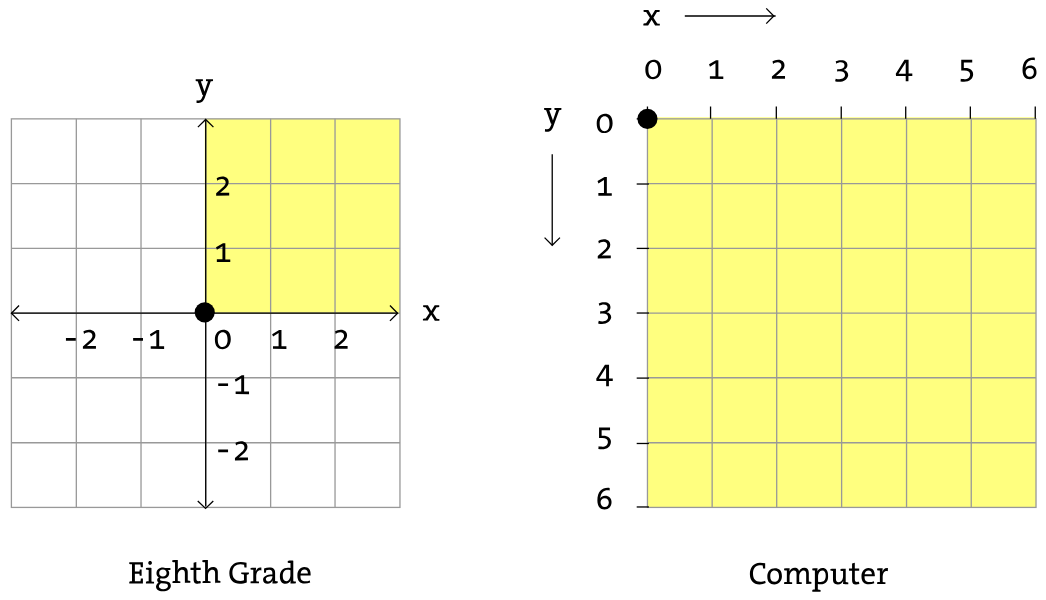
line(1,0,4,5);

Навіть без вивчення синтаксису написання коду, наведене вище твердження повинно мати достатній сенс. Ми надаємо команду (яку ми будемо називати «функцією»), щоб машина слідувала під назвою «рядок». Крім того, ми вказуємо деякі аргументи щодо того, як ця лінія має бути проведена від точки A (1,0) до точки B (4,5). Якщо ви розглядаєте цей рядок коду як речення, функція є дієсловом, а аргументи є об’єктами речення. Кодове речення також закінчується крапкою з комою замість крапки.



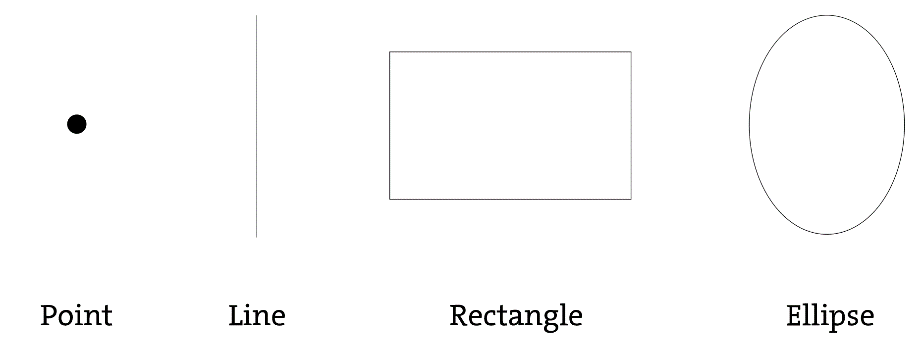
Ключовим тут є усвідомлення того, що екран комп’ютера – це не що інше, як красивіший шматок міліметрового паперу. Кожен піксель екрана — це координата — два числа, «x» (горизонталь) і «y» (вертикаль), які визначають розташування точки в просторі. І наша робота полягає в тому, щоб визначити, які форми та кольори мають з’являтися в цих піксельних координатах.

Тим не менш, тут є підступ - ("декартова система координат"), розміщений (0,0) у центрі з віссю y, спрямованою вгору, а віссю x, спрямованою праворуч (у позитивному напрямку, негативному вниз і ліворуч ). Однак система координат для пікселів у вікні комп’ютера змінюється на протилежну вздовж осі y. (0,0) можна знайти у верхньому лівому куті з позитивним напрямком праворуч по горизонталі та вниз по вертикалі.



**Прості фігури**

Переважна більшість прикладів програмування p5 є візуальними. Ці приклади, за своєю суттю, включають малювання фігур і встановлення пікселів. Почнемо з розгляду чотирьох примітивних форм.



Для кожної форми ми запитаємо себе, яка інформація потрібна, щоб визначити розташування та розмір (а пізніше колір) цієї форми, і дізнаємось, як p5 очікує отримати цю інформацію. На кожній із наведених нижче діаграм ми припустимо вікно шириною 100 пікселів і висотою 100 пікселів.

Point [()](https://p5js.org/reference/#/p5/point) — це найпростіша з форм, з якої можна почати. Щоб намалювати точку, нам потрібні лише координати x і y.

function setup(){

createCanvas(100, 100);

}

function draw(){

point(40, 50); // point(x, y)

}

Функція [line()](https://p5js.org/reference/" \l "/p5/line) теж не надто складна і вимагає лише двох точок: (x1,y1) і (x2,y2):

function setup(){

createCanvas(100, 100);

}

function draw(){

line(10, 20, 50, 20); // line(x1, y1, x2, y2)

}

Коли ми прийшли до малювання [rect()](https://p5js.org/reference/" \l "/p5/rect) , все стає дещо складнішим. У p5 прямокутник визначається координатою верхнього лівого кута прямокутника, а також його шириною та висотою.

function setup(){

createCanvas(100, 100);

}

function draw(){

rect(10, 20, 40, 30); // rect(x, y, width, height)

}

Другий спосіб намалювати прямокутник включає в себе визначення центральної точки разом із шириною та висотою. Якщо ми віддаємо перевагу цьому методу, ми спочатку вказуємо, що ми хочемо використовувати режим [CENTER](https://p5js.org/reference/#/p5/CENTER) перед інструкцією для самого прямокутника. Зауважте, що p5 чутливий до регістру.

function setup(){

createCanvas(100, 100);

rectMode(CENTER);

}

function draw(){

rect(30, 20, 40, 20); // rect(x, y, width, height)

}

Нарешті, ми також можемо намалювати прямокутник з двома точками (верхній лівий кут і нижній правий кут). Режим тут [КУТИ](https://p5js.org/reference/#/p5/CORNERS) . Зауважте, що цей приклад дає той самий результат на екрані, що й приклад вище.

function setup(){

createCanvas(100, 100);

rectMode(CORNERS);

}

function draw(){

rect(10, 10, 50, 30); // rect(x1, y1, x2, y2)

}

Коли ми освоїмося з концепцією малювання прямокутника, [ellipse()](https://p5js.org/reference/" \l "/p5/ellipse) стане нескладним. Фактично, це ідентично [rect()](https://p5js.org/reference/" \l "/p5/rect) з тією різницею, що еліпс малюється там, де буде обмежувальна рамка прямокутника. Типовим режимом для [ellipse()](https://p5js.org/reference/" \l "/p5/ellipse) є [CENTER](https://p5js.org/reference/#/p5/CENTER) , а не [CORNER](https://p5js.org/reference/#/p5/CORNER) .

  function setup(){

createCanvas(100, 100);

ellipseMode(CENTER);

}

function draw(){

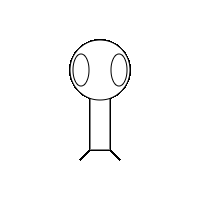
ellipse(30, 30, 40, 60); // ellipse(x, y, width, height)

}

Тепер давайте подивимося на код із фігурами в більш повній формі з розмірами полотна 200 на 200. Зверніть увагу на використання функції createCanvas() для визначення ширини та висоти полотна.

Завдання

-намалюйте в програмі отакий графічний примітив



**Координати і перетворення**

WebGL — це бібліотека, яка надає нам інструменти, необхідні для створення 3D-графіки у веб-переглядачі. Простіше кажучи, це дозволяє нам виконувати всі види математики, щоб допомогти впорядкувати та відобразити об’єкти в 3D. p5.js полегшує роботу з WebGL завдяки спеціальному режиму WebGL.

Робота в 3D дуже складна, особливо коли ескіз містить рух, текстуру, освітлення тощо. На щастя для нас, комп’ютери мають спеціальне апаратне забезпечення, яке особливо добре підходить для виконання цих обчислень, графічний процесор (GPU). Графічний процесор здатний обробляти багато речей одночасно, що особливо важливо, коли ми маємо справу з пікселями та багатьма фігурами в просторі.

Давайте почнемо з налаштування p5.js для використання WebGL, передавши WEBGL як третій параметр у createCanvas() .

function setup() {

createCanvas(windowWidth, windowHeight, WEBGL);

describe('a red box on a white background');

}

function draw(){

background(255);

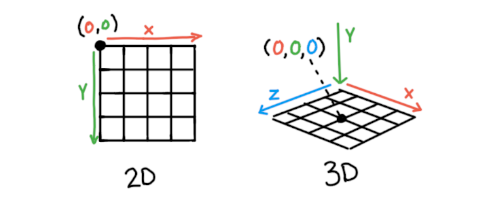
fill(255,0,0);

box();

}

Одна з найбільш фундаментальних відмінностей між роботою в 2D і роботою в 3D є найбільш очевидною: є ще один вимір для роботи. На додаток до горизонтального та вертикального положення (осі x і y) елемента на нашому малюнку, 3D додає глибину, вісь z.

Під час малювання у 2D точка (0,0) розташована у верхньому лівому куті екрана. У режимі WebGL вихідна точка ескізу (0,0,0) розташована посередині екрана. За замовчуванням вісь x йде зліва направо, вісь y - вгору-вниз, а вісь z - далі-наближче.



Трансформації: положення та розмір тривимірних фігур

У p5.js є кілька методів translate() , rotate() і scale() , які ми можемо використовувати для розташування та орієнтації об’єктів у просторі. Кожен із цих методів впливає на те, що називається матрицею моделі . Разом вони відомі як перетворення об’єкта. Ці методи доступні як для 2D, так і для 3D малювання.

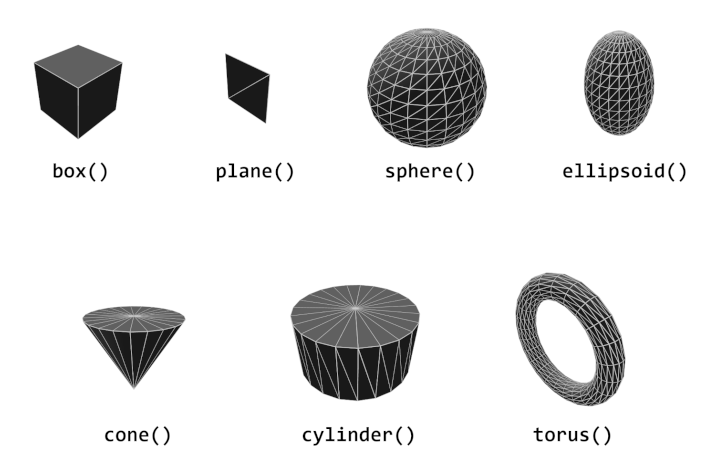
translate() переміщує початок координат у заданому напрямку. Все, що намальовано після виклику translate(), буде розташовано відносно цієї точки. translate() приймає аргументи для значень x, y та z. Використовуйте повзунки на наведеному вище ескізі, щоб змінити переклад прямокутника та побачити, як він рухається вздовж кожної осі. Наведений нижче код демонструє простий переклад форми box() .

rotate() переорієнтує все, що намальовано після нього.

Існує кілька методів, за допомогою яких можна обертати об’єкт у 3D. У більшості випадків найпростіше викликати такі методи, як rotateX() , rotateY() і rotateZ() , кожен з яких дозволяє обертати навколо певної осі. Кожен із цих методів приймає один аргумент, який визначає кут повороту. Спробуйте перемістити повзунки у прикладі вище, щоб побачити, як виконується обертання на кожній осі. Наведений нижче код показує, що кожен із цих методів використовується.

Основні форми в 3D

Поки що ми використовували лише box(), але p5.js має сім різних попередньо визначених геометрій, які ви можете використовувати у своєму ескізі. Ці базові попередньо визначені геометрії часто називають «примітивами».



Дані миші

Змінні mouseX і mouseY (зверніть увагу на великі букви X і Y) зберігають координату x і y курсора відносно початку координат у верхньому лівому куті вікна дисплея. Щоб побачити фактичні значення, отримані під час переміщення миші, запустіть цю програму, щоб вивести значення на екран:

function draw() {

background(255);

frameRate(12);

text("X: "+mouseX, 0, height/4);

text("Y: "+mouseY, 0, height/2);

}

Коли програма запускається, значення mouseX і mouseY дорівнюють 0. Якщо курсор переміщується у вікно дисплея, значення встановлюються на поточну позицію курсору. Якщо курсор знаходиться ліворуч, значення mouseX дорівнює 0 і значення збільшується, коли курсор переміщується праворуч. Якщо курсор знаходиться вгорі, значення mouseY дорівнює 0 і значення збільшується, коли курсор рухається вниз. Якщо mouseX і mouseY використовуються в програмах без draw() або якщо noLoop() запускається в setup() , значення завжди будуть 0.

Положення миші найчастіше використовується для керування розташуванням візуальних елементів на екрані. Більш цікаві зв’язки створюються, коли візуальні елементи по-різному співвідносяться зі значеннями миші, а не просто імітують поточну позицію. Додавання та віднімання значень від положення миші створює зв’язки, які залишаються постійними, а множення та ділення цих значень створює змінні візуальні зв’язки між положенням миші та елементами на екрані. У першому з наведених нижче прикладів коло безпосередньо відображається на курсорі, у другому числа додаються та віднімаються від позиції курсора для створення зсувів, а в третьому множення та ділення використовуються для масштабування зміщень.

function setup() {

createCanvas(100, 100);

noStroke();

}

function draw() {

let x = mouseX;

let y = mouseY;

let ix = width - mouseX; // Inverse X

let iy = height - mouseY; // Inverse Y

background(126);

fill(255, 150);

ellipse(x, height/2, y, y);

fill(0, 159);

ellipse(ix, height/2, iy, iy);

}

Геометрію також можна визначити процедурно за допомогою коду. Це чудовий спосіб створити фігуру, яка рухається або формується за вашим власним набором правил. Існує кілька методів, які можна використовувати для створення 3D-геометрії у спосіб, подібний до 2D-малюнків у p5.js. Наприклад, такі методи, як quad() , triangle() , rect() і circle() , мають додаткові параметри, які дозволяють використовувати їх у 3D.

Існують інші методи, які забезпечують більший контроль над геометрією. Фігуру можна визначити точково за допомогою beginShape() , vertex() і endShape(). Цей наступний приклад показує, як ці методи можна використовувати для математичної побудови тривимірної форми.

function setup() {

createCanvas(windowWidth, windowHeight, WEBGL);

describe('a red 3d spiral shape created with beginShape and endShape');

}

function draw(){

background(220);

// click+drag on canvas to look around the shape

orbitControl();

fill(255,0,0,128);

var numVertices = 100;

translate(0,numVertices,0);

rotateY(millis()/1000);

scale(20);

beginShape();

// a simple spiral shape

for(let i = 0; i < numVertices; i++) {

vertex(

sin((i/numVertices\*TWO\_PI)\*5)\*sin(i/numVertices\*PI)\*2,

-i/10,

cos((i/numVertices\*TWO\_PI)\*5)\*sin(i/numVertices\*PI)\*2,

);

}

endShape();

}

Завдання

-Використовуйте змінні mouseX і mouseY зі структурою if, щоб дозволити курсору вибирати області екрана. Перший ділить екран на половини, а другий ділить екран на третини.

-Використовуючи код синього кольору змінити напрямок руху, колір а також фігуру, що рухається.