МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ

Кафедра «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Мерката»

Затверджено
НМК Інституту інформаційних технологій та інноваційного підприємництва
Протокол № 3 від 04.12.2025
Керівник НМК

_____ Андрій ІВАНОВ

"4" грудня _____ 2025 р.

Методичні вказівки
до лабораторних робіт
з дисципліни «Основи концепції інтернет речей».
Частина 2. Розробка інтерфейсів у системах інтернету речей. Програмування Processing.

Для здобувачів вищої освіти спеціальності
F3 «Комп'ютерні науки» першого (бакалаврського) рівня
для денної та заочної форми навчання.

Одеса 2025

Методичні вказівки до лабораторних робіт з дисципліни «Основи концепції інтернет речей» Частина 2. Розробка інтерфейсів інтернет-речей. Програмування Processing підготовлені Ларіним Дмитром Георгійовичем, к.т.н., доцентом кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта» Одеського національного морського університету та Тузовим Олександром Валерійовичем, старшим викладачем кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта» Одеського національного морського університету.

Методичні вказівки до лабораторних Робіт з дисципліни «Основи концепції інтернет речей» схвалено кафедрою «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта» ОНМУ

«_06_»_____11_____ 2025 року, протокол №_8_

Завідувач кафедрою _____ Павло НОСОВ

Зміст

Лабораторна робота «Ознайомлення з мовою програмування Processing»

Теоретичні положення

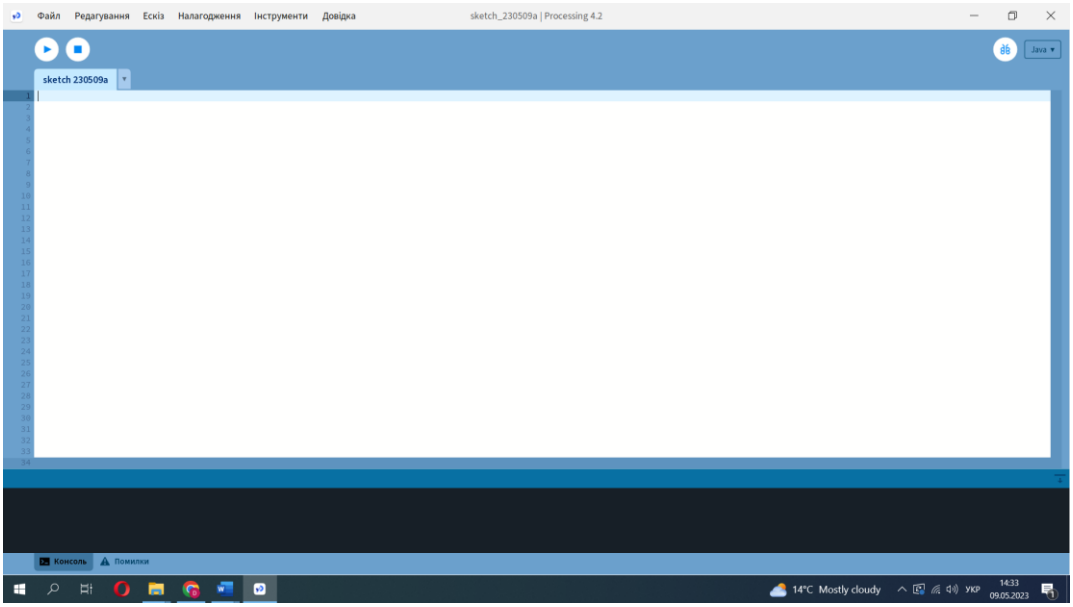
Processing - це відкрита мова програмування, заснована на java з простим і зрозумілим сі-подібним синтаксисом. Являє собою легкий і швидкий інструментарій для людей, які хочуть програмувати зображення, анімацію і інтерфейси.

В принципі, також можна створювати навіть 3D-аплікації (в тому числі і ігри), адже processing має засоби підтримки OpenGL. Всі ці можливості, в сукупності з великою кількістю функцій і дуже логічним синтаксисом, роблять цю мову ідеальним для навчання і прищеплення інтересу до програмування.

Є інструменти для побудови графічних примітивів, 3D-об'єктів, робота зі світлом, текстом, інструментами трансформації. Також ми можемо імпортувати і експортувати файли аудіо / відео / звукових форматів, обробляти події миші / клавіатури, працювати зі сторонніми бібліотеками (openGL, PDF, DXF), працювати з мережею.

Запускаємо файл під назвою processing.exe

Перед нами відкривається ось таке вікно:



Також, як і в Arduino IDE, в Processing'е є дві основні функції:

* setup()

* draw () (такі ж функції як і в loop())

Приклад

Нам потрібні наступні компоненти**:**

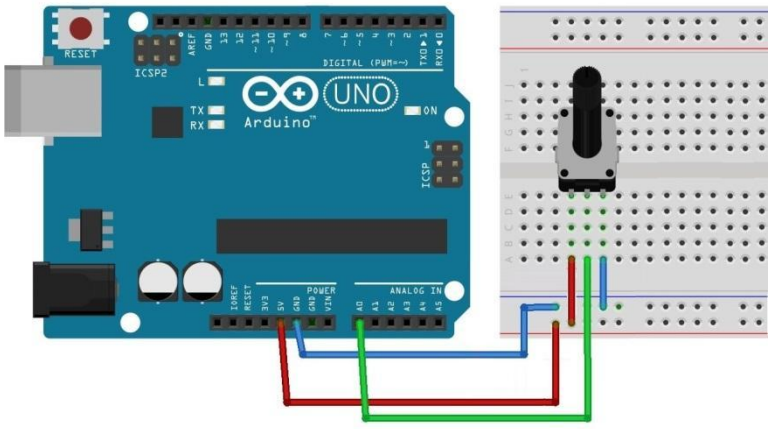
* Arduino Uno (або інший Arduino контролер)

* Макетна плата

• З'єднувальні дроти "папа-папа"

* Потенціометр (в роботі використовується на 10кОм)

Для початку збираємо ось таку схему:



Тепер відкриваємо Arduino IDE і вставляємо наступний скетч:

```
int potPin = 0; // пін A0 до якого под’єднаний потенціометр
void setup()
{
  Serial.begin (9600); // встановлюємо швидкість передавання по
//COM порту
}
void loop()
{
 int dat = map(analogRead(potPin),0,1023,0,255); //перетворюємо
// сигнал потенціометру з діапазону 0-1023 в діапазон 0-255
 Serial.println(dat); //передаємо дані в COM порт
 delay(50);
}
```

У цьому коді зчитується значення з аналогового входу A0 (potPin). І це значення записується в цифровому діапазоні в змінну *dat*. Яку потім ми відправляємо в послідовний порт.

Якщо потенціометр приєднаний до іншого входу контролера – виправляємо код.

Тепер це значення потрібно отримати і обробити. Для цього заходимо в середовище розробки Processing **(**PDE). І пишемо наступний скетч:

```
import processing.serial.*;
Serial port;
float dat = 0;

void setup()
{
 size(200,200);
 port = new Serial(this,"COM5",9600);
 port.bufferUntil('\n');
}

void draw()
{
 background(0,0,dat);
```

```
}

void serialEvent(Serial port)
{
 dat = float (port.readStringUntil ('\n'));
}
```

Перед тим як запустити, більш детально з кодом.

Імпортуємо Serial бібліотеку для роботи з послідовним COM портом

```
import processing.serial.*;
```

Створюємо об'єкт Serial під назвою port

```
Serial port;
```

Створюємо змінну dat типу float. Це значення, що ми будемо отримувати з Arduino. Те саме значення, яке змінюється зі зміною опору потенціометра

```
float dat = 0;
```

Так само як і в Arduino IDE, в Processing'e є функція setup() . Насамперед, в цій функції, треба створити наше робоче вікно. Створюємо вікно з розмірами 200x200 пікселів

```
size(200,200);
```

Ініціалізуємо порт. Також як в Arduino - Serial.begin. Тільки трохи інакше. У прикладі Arduino підключено до порту "COM5" (у Вас цей порт може бути іншим).

```
port = new Serial(this,"COM5",9600);
```

При роботі в ОС Linux команда може мати вигляд *port = new Serial(this, "/dev/ttyACM0", 9600);*

Тепер перевіряємо, чи надходить на наш порт що-небудь. ('\n') - символ, що означає переклад на новий рядок.

```
port.bufferUntil('\n');
```

Функція drow () (як і loop () в Arduino IDE), працює як нескінченний цикл оновлення екрану. Функція background (R,G,B), де R-червоний, G-зелений, B-синій кольори. змінює фон створеного нами вікна. В даному

випадку, ми будемо міняти тільки синій колір, від чорного, до яскраво-синього.

```
background(0,0,dat);
```

Наступна функція serialEvent (Serial port). Ця функція викликається автоматично, при надходженні нових даних на порт, грунтуючись на buferUntil який ми вказали у функції setup ().

Тепер читаємо значення, яке приходить у порт, перетворюємо його до типу з плаваючою точкою, і записуємо його в змінну dat. Ми зчитуємо повністю весь рядок, до приходу символу '\n'.

```
dat = float (port.readStringUntil ('\n'));
```

Експеримент 1

Додамо ще два потенціометра, кожен з яких буде відповідати за свій колір. А в вікно програми додамо кілька квадратів, відповідні потенціометрам, будемо міняти їх колір, змінюючи значення опорів резисторів. А також міняти колір фону.

Що нам потрібно для експерименту:

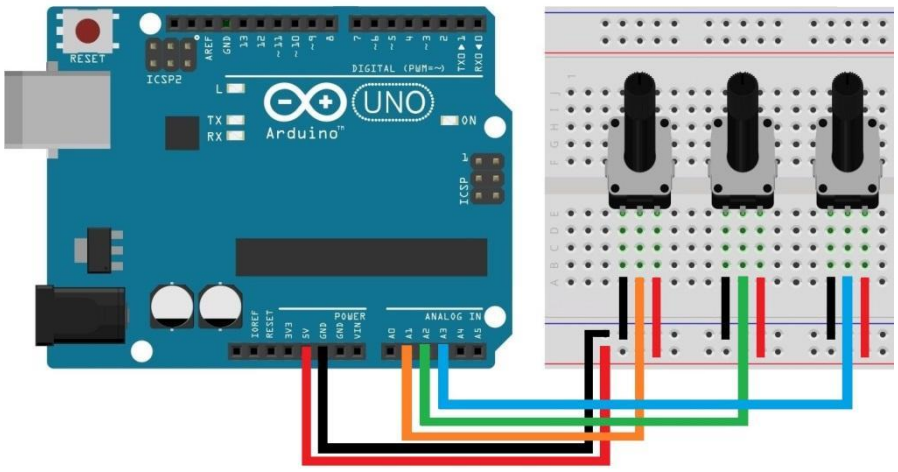
* Arduino Uno (або інший Arduino контролер)

* Макетна плата

• З'єднувальні дроти "папа-папа"

• 3 змінних резистора (використовується на 10кОм)

Збираємо ось таку схему:



Тепер треба змінити прошивку Arduino. Для цього потрібно вставити ось такий скетч в Arduino IDE:

```
int R = A1;
int G = A2;
int B = A3;

void setup()
{
  Serial.begin (9600);
}

void loop()
{
 int dat_1 = map(analogRead(R),0,1023,0,255);
 Serial.println(dat_1);
 delay(50);
 int dat_2 = map(analogRead(G),0,1023,256,511);
 Serial.println(dat_2);
 delay(50);
 int dat_3 = map(analogRead(B),0,1023,512,767);
 Serial.println(dat_3);
 delay(50);
}
```

Є три змінні, що відповідають кожна за порт свого потенціометра (R,G,B). Ми читаємо значення і відправляємо. На приймальній стороні нам потрібно зрозуміти, яке зі значень відповідає якому кольору. Для цього розбиваємо діапазон значень.

Наприклад, для червоного кольору, значення будуть коливатися від 0 до 255. У зеленого кольору, від 256 до 511. І у синього від 512 до 767. Тобто на кожен колір 256 значень.

Тепер відкриваємо PDE і вставляємо цей код:

```
import processing.serial.*;
Serial port;
```

```
float R = 0;
float G = 0;
float B = 0;

void setup()
{
 size (170,170);
 port = new Serial(this, "COM5",9600);
 port.bufferUntil('\n');
}

void draw()
{
 background(R,G,B);
 fill(R,0,0);
 rect(10,10,50,50);
 fill(0,G,0);
 rect(60,60,50,50);
 fill(0,0,B);
 rect(110,110,50,50);
}

void serialEvent (Serial port)
{
 float dat = float (port.readStringUntil ('\n'));

 if ( dat >= 0 && dat <=255)
 {
 R = dat;
 }
 else if (dat >= 256 && dat <=511)
 {
 G = map(dat,256,511,0,255);
 }
```

```
 else if (dat >= 512 && dat <=767)
 {
 B = map(dat,512,767,0,255);
 }
}
```

Змінилася умова прийому значень у функції serialEvent. Наприклад, прийшло значення. Записуємо його в змінну dat. Тепер, якщо це значення від 0 до 255, значить це прийшло значення червоного кольору. Записуємо його в змінну R. якщо приходять значення від 256 до 511, то програма розуміє, що прийшли значення зеленого кольору. Зіставляє їх з цифровим діапазоном, і записує в змінну G. теж саме і з синім кольором.

Тепер, коли прийняли значення червоного, зеленого і синього кольорів, потрібно ці значення застосувати. Все це відбувається у функції drow ().

Змінює фон нашого вікна, на колір, який є сумою всіх трьох кольорів.

```
background(R,G,B);
```

Тепер додаємо квадрат с допомогою функції rect (x, y, a, b), де x - початкова координата по x осі, y - початкова координата по y осі, a - ширина квадрата (в пікселях), b - висота квадрата. Початкові координати відраховуються від верхнього лівого кута. Тобто в верхньому лівому кутку координати x і y рівні 0.

Квадрат буде мати заливку, описану в функції fill (R, G, B)**.**

```
fill(R,0,0);
rect(10,10,50,50);
```

За аналогією додаємо ще два квадрата. Зі своїми координатами і заливкою.

Загалом весь принцип такий: є три квадрата, що відповідають за свій колір. А фон вікна залежить від суми цих трьох кольорів.

Запустіть програму.

Експеримент 2

Давайте створимо невелику експозицію, що складається з тих же квадратів і потенціометрів. Функції і схема залишаться тими ж самими, але трохи попрацюємо з самої графічної середовищем.

Міняти код Arduino не будемо. Змінимо тільки скетч Processing'a.

```
import processing.serial.*;
Serial port;

float R = 0;
float G = 0;
float B = 0;

void setup()
{
 size (300,300);
 port = new Serial(this, "COM5",9600);
 port.bufferUntil('\n');
}

void draw()
{
  fill(R,0,0);
 rect(0,0,300,300);

 fill(0,G,0);
 rect(15,15,270,270);

 fill(0,0,B);
 rect(30,30,240,240);

 fill(R,0,0);
 rect(45,45,210,210);

 fill(0,G,0);
 rect(60,60,180,180);
```

```
 fill(0,0,B);
 rect(75,75,150,150);

 fill(R,0,0);
 rect(90,90,120,120);

 fill(0,G,0);
 rect(105,105,90,90);

 fill(0,0,B);
 rect(120,120,60,60);

 fill(R,G,B);
 rect(135,135,30,30);
}

void serialEvent (Serial port)
{
  float dat = float (port.readStringUntil ('\n'));

 if ( dat >= 0 && dat <=255)
 {
 R = dat;
 }
 else if (dat >= 256 && dat <=511)
 {
 G = map(dat,256,511,0,255);
 }
 else if (dat >= 512 && dat <=767)
 {
 B = map(dat,512,767,0,255);
 }
}
```

Додано 10 квадратів, різних розмірів і кольорів. Останній квадрат має найменші розміри і його заливкою буде сума всіх кольорів.

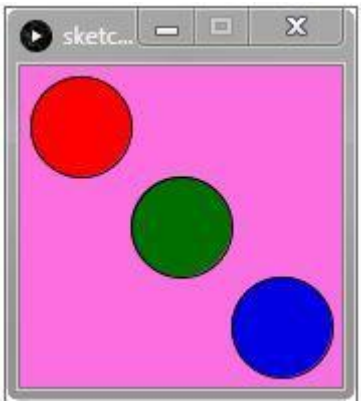
Запустіть програму.

Виконання

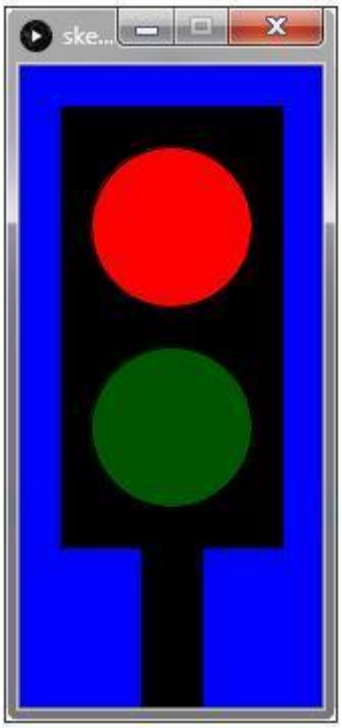
1. Ознайомитися з інтерфейсом ***Processing***.

2. Зібрати наведені експерименти 1 та 2.

3. Змініть програму, щоб при зміні опорів потенціометрів, змінювалася заливка не квадратів, а кіл. А також заливка фону була сумою всіх кольорів. Функція для кола ellipse (x,y,a,b); де x і y - координати центру кола. А a і b- діаметри по осі X і Y відповідно. Ось щоб вийшло щось типу цього.



4. Зробіть невеликий світлофор з 2 кольорів: червоного і зеленого. А синій колір використовувати як фон. Повинно бути приблизно так:



5. Продемонструйте результати викладачу.

Лабораторна робота «Створення фігур у Processing»

Щоб створювати красиві інтерфейси, потрібно вміти малювати фігури.

Почнемо з основ. Як вже було сказано раніше - нульова точка вікна в Processing знаходиться у верхньому лівому куті. Функція size() встановлює розміри вікна вашого ескізу. Перший параметр присвоює значення вбудованої змінної width (ширина), другий - вбудованої змінної height (висота).

Для простих прикладів поки тимчасово не будемо використовувати функцію draw (), а весь код будемо писати в функції setup ().

Для малювання точки використовується функція point (). Тут також потрібно вказати два параметри - координати точки.

Створимо вікно розміром 480 на 120 і помістили точку в центрі (розділимо розміри вікна навпіл).

```
void setup() {
size(480, 120);
point(240, 60);
}
```

Точка дуже маленька, розгледіти її складно. Якщо ви хочете намалювати точку в нижньому правому куті, то можете зробити помилку, написавши код.

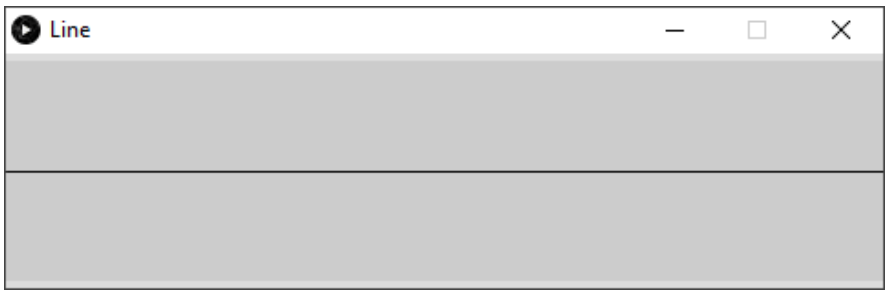
```
point(480, 120);
```

Насправді, потрібно використовувати координати (479, 119), тобто відняти одиницю від розмірів вікна.

Щоб намалювати лінію, потрібно викликати функцію line() с чотирма параметрами - координати початкової і кінцевої точки. А програма сама намалює лінію між ними. Розділимо вікно програми навпіл. Видалимо код для малювання точки, а замість неї надрукуємо інший код.

```
void setup() {
size(480, 120);
line(0, 60, 480, 60);
}
```

Отримаємо наступну картинку.

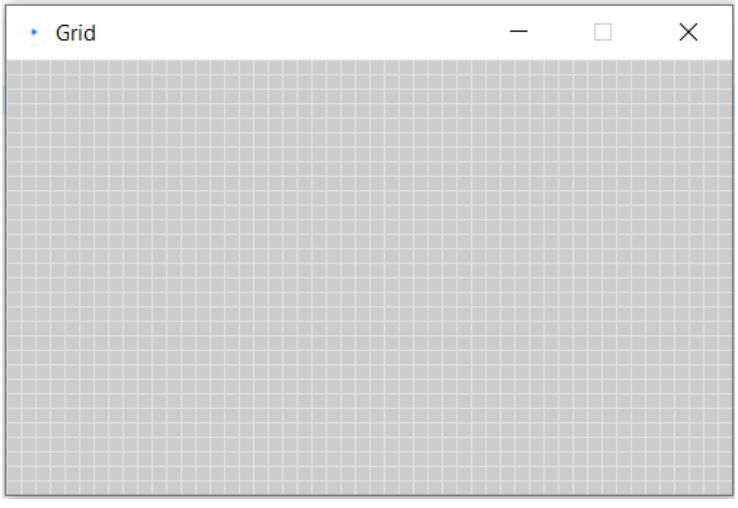


Спробуйте намалювати вертикальну лінію та лінію по діагоналі.

Якщо необхідно намалювати багато паралельних ліній по вертикалі і горизонталі через однакові проміжки - тоді отримаємо сітку. Щоб не

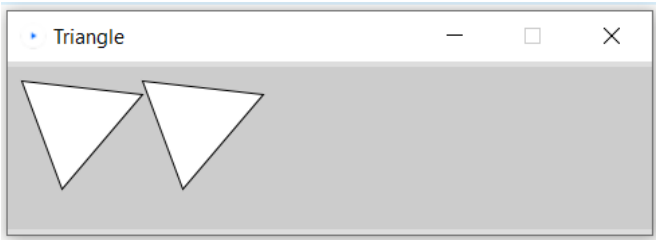
повторювати один і той же код багато разів, створимо окрему функцію для малювання сітки, а в ній застосуємо два циклу. Код вийде набагато коротше.

```
void setup()
{
  size(500, 300);
}

void draw()
{
  drawGrid();
}

void drawGrid()
{
  stroke( 225 );
  for ( int i = 0; i < 64; i++ ) {
    line(i * 10, 0, i * 10, height );
  }
  for ( int i = 0;
    i < 48;
    i++ ) {
    line( 0, i * 10, width, i * 10 );
  }
}
```



Спробуємо намалювати трикутник. Для цього існує функція triangle() з шістьма параметрами - координати вершин трикутника. Навмання вибираємо різні числа. Вийшов трикутник. Захотілося намалювати поруч близнюка. Як це зробити? Щоб зрушити другий трикутник вправо, додаємо до параметрів, які відносяться до координати X, деяке значення. Це буде зміщенням.

```
void setup()
{
  size(480, 120);
  triangle(10, 10, 100, 20, 40, 90);
  // Переміщуємо другий трикутник праворуч на 90 пикселей
  triangle(10 + 90, 10, 100 + 90, 20, 40 + 90, 90);
}
void draw(){ }
```

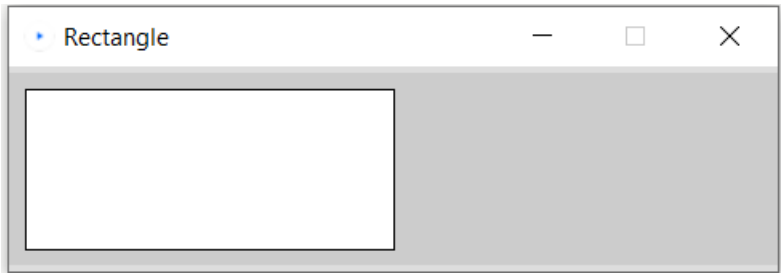


Для багатокутників з чотирма точками потрібно вже вісім параметрів. А функція називається quad ().

```
void setup()
{
  size(480, 120);
  quad(20, 10, 30, 40, 90, 25, 20, 90);
}
void draw(){}
```

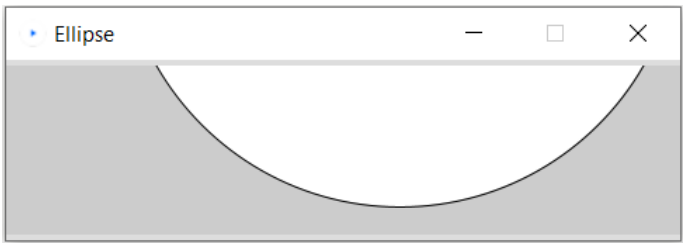
Для того, щоб намалювати прямокутник, з попередньої роботи пам’ятаємо, що використовується функція rect().

```
void setup()
{
  size(480, 120);
  rect(10, 10, 230, 100);
}
void draw(){}
```



Функція ellipse() також вже знайома по попередній роботі. Для малювання еліпса потрібно вказати центр, ширину і висоту. Варто відзначити, що можна вказувати і негативні значення координат. Тоді може вийти, що у вікні ви побачите тільки частина фігури. Наприклад, частина еліпса.

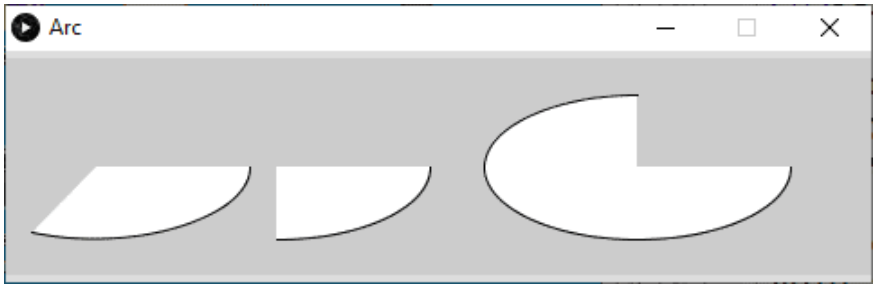
```
void setup()
{
  size(480, 120);
  ellipse(280, -100, 400, 400);
}

void draw(){}
```



Також можна намалювати дугу (сектор) за допомогою функції arc(). Вказуємо центр, а також ширину і висоту. до того ж вказуємо початковий і кінцевий кут в радіанах. Але з радіанами працювати не дуже зручно. Для

величин 180, 45, 90 і 360 градусів можна використовувати готові константи в радіанах: PI, QUARTER_PI, HALF_PI, TWO_PI. Спробуємо кілька варіантів.

```
void setup() {
  size(480, 120);
  arc(50, 60, 170, 80, 0, 2);
  arc(150, 60, 170, 80, 0, HALF_PI);
  arc(350, 60, 170, 80, 0, PI + HALF_PI);
}
```



Якщо відомо значення в градусах, то можете його перевести в радіани за допомогою функції radians().

```
arc(90, 60, 80, 80, 0, radians(90));
```

Якщо малюєте багато фігур, які накладаються одна на одну, то зверху буде та фігура, чий код у програмі буде останнім.

У фігур можна управляти товщиною лінії і режимом згладжування. За замовчуванням, товщина ліній становить один піксель. За допомогою функції strokeWeight() можна це змінити. Товщину слід встановлювати до виведення фігури. Якщо малюєте кілька фігур, то у кожної треба встановлювати товщину ліній окремо. Інакше у фігури буде товщина, задана для попередньої фігури.

```
strokeWeight(3);
ellipse(175, 60, 90, 90);
```

Функція strokeJoin() визначає вид з'єднання ліній (кути), а функція strokeCap() - початкову і кінцеву точки ліній.

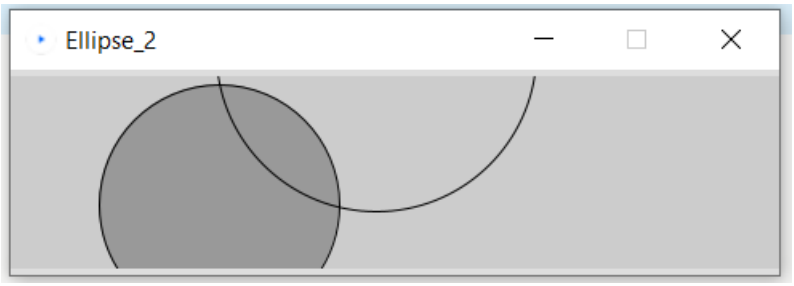
```
strokeJoin (ROUND); / / скруглити
strokeJoin (BEVEL); / / зробити укіс
```

```
strokeCap (SQUARE); / / кінці ліній-квадратні
strokeCap (ROUND); / / округлити кінці ліній
```

За згладжування відповідає функція smooth(). Існує парна їй функція noSmooth (), що відключає згладжування.

Для коьорів використовуються функції background(), fill(), stroke(), у яких потрібно вказати колір в межах від 0 (чорний) до 255 (білий). Проміжні величини будуть відтінками сірого. Можна відключити обведення (функція noStroke()) або зробити фігуру прозорою функцією noFill():

```
void setup() {
size(480, 120);
smooth();
fill(153);
ellipse(130, 80, 150, 150);
noFill();
ellipse(228, -16, 200, 200);
noStroke();
ellipse(268, 118, 200, 200);
}
```



Також можна вказувати колір. У цьому випадку використовуються три параметри:

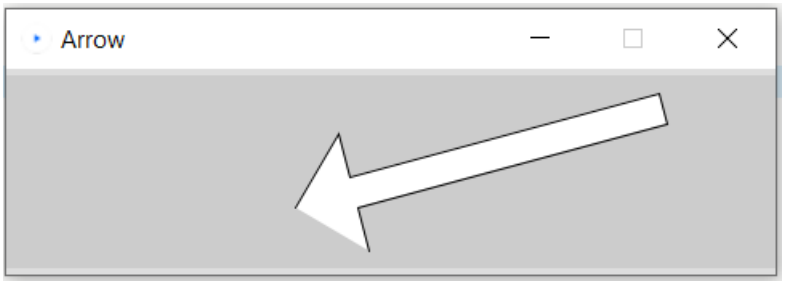
```
background(0, 26, 51); // темно-синій колір
fill(255, 0, 0); // червоний
```

Перша функція зафарбує вікно, друга - фігуру, яка буде малюватися після цієї функції.

Можна використовувати колірну палітру через меню Tools | Color Selector. Якщо додати четвертий параметр, то можна управляти прозорістю.

Також можна намалювати складну фігуру, наприклад, стрілку. Почніть створення фігури з функції beginShape(). Функція vertex() використовується для визначення x і y -координат фігури. Функція endShape() ставиться в кінці опису фігури і сигналізує про закінчення створення фігури.

```
void setup() {
size(480, 120);
smooth();
beginShape();
vertex(180, 82);
vertex(207, 36);
vertex(214, 63);
vertex(407, 11);
vertex(412, 30);
vertex(219, 82);
vertex(226, 109);
endShape();
}
```



Коли ви запустите приклад, ви побачите, що перша і остання точки не з'єднані. Щоб зробити це, додайте слово CLOSE як параметр функції endShape():

```
endShape(CLOSE);
```

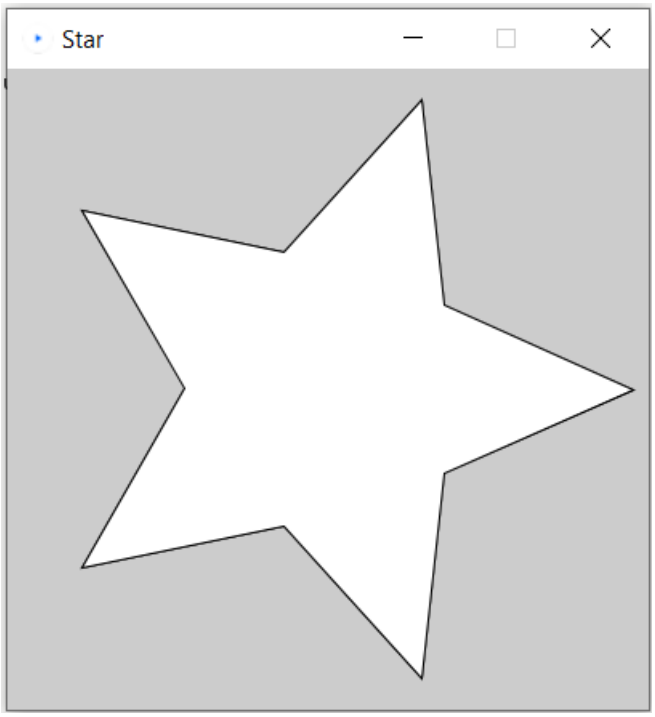
Перевага використання функції vertex () для створення фігур полягає в можливості побудови фігур зі складною структурою. Одна програма на Processing може намалювати тисячі ліній для відображення на екрані найнеймовірніших фігур.

Зірка і квітка

Малюємо зірку. Для неї створити окрему функцію zirka().

```
void setup()
{
size( 400, 400 );
smooth();
frameRate(1);
}
void draw() {
translate(200, 200);
zirka(5, 90, 190);
}
void zirka( int numSpikes, float innerRadius, float outerRadius
)
{
int numVertices = numSpikes * 2;
float angleStep = TWO_PI / numVertices;
beginShape();

for ( int i = 0; i < numVertices; i++ ) {
float x, y;
if ( i % 2 == 0 ) {
x = cos( angleStep * i ) * outerRadius;
y = sin( angleStep * i ) * outerRadius;
} else {
x = cos( angleStep * i ) * innerRadius;
y = sin( angleStep * i ) * innerRadius;
}
vertex( x, y );
}
endShape(CLOSE);
}
```

Функція зірки має три параметри: одне ціле число для кількості променів і два параметри для внутрішнього і зовнішнього радіуса зірки.

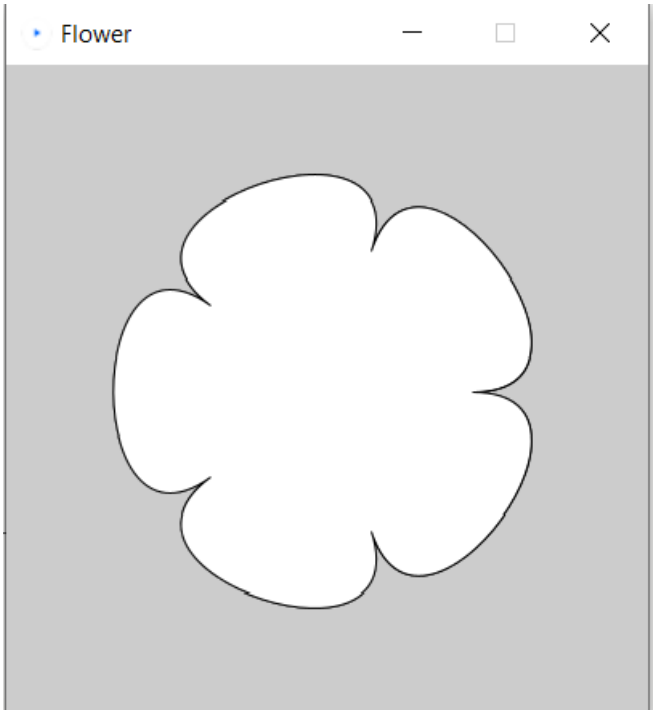


Додамо функцію для малювання квітки.

```
void kvitka( int numLeafs, float innerRadius, float outerRadius
)
{
float angleStep = TWO_PI / numLeafs;
beginShape();
float startX = cos( 0 ) * innerRadius;
float startY = sin( 0 ) * outerRadius;
vertex( startX, startY );
for ( int i = 0; i < numLeafs; i++ ) {
float cx1 = cos( angleStep * i ) * outerRadius;
float cy1 = sin( angleStep * i ) * outerRadius;
float x2 = cos( angleStep * (i + 1) ) * innerRadius;
float y2 = sin( angleStep * (i + 1) ) * innerRadius;
float cx2 = cos( angleStep * (i + 1) ) * outerRadius;
float cy2 = sin( angleStep * (i + 1) ) * outerRadius;
bezierVertex( cx1, cy1, cx2, cy2, x2, y2 );
}
endShape( CLOSE );
}
```

Викличемо замість зірки.

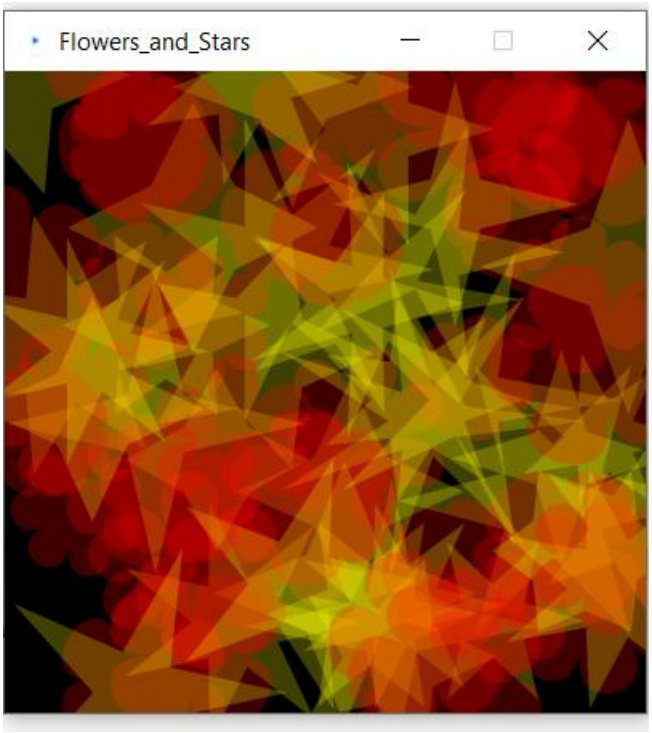
```
void draw() {
  translate(200, 200);
  kvitka(5, 90, 190);
}
```



А тепер намалюємо зірки і квіти разом.

```
void draw() {
  background( 0 );
  noStroke();
  for ( int i = 0; i < 75; i++ ) {
    int numPoints = floor( random( 4, 8 ) );
    float innerRadius = random( 20, 40 );
    float outerRadius = random( 50, 100 );
    pushMatrix();
    translate( random( width ), random( height ) );
    if ( random( 100 ) < 50 ) {
      fill( 255, 255, 0, 64 );
      zirka( numPoints, innerRadius, outerRadius );
    } else {
      fill( 255, 0, 0, 64 );
      kvitka( numPoints, innerRadius, outerRadius );
    }
```

```
    popMatrix();
  }
}
```



Виконання

1. Ознайомитися з функціями малювання фігур.
2. Створити функцію власної фігури (багатокутника). Параметри – кількість кутів, колір, розмір.
3. Продемонструйте результати викладачу.

Лабораторна работа «Динамічні інтерфейси в Processing»

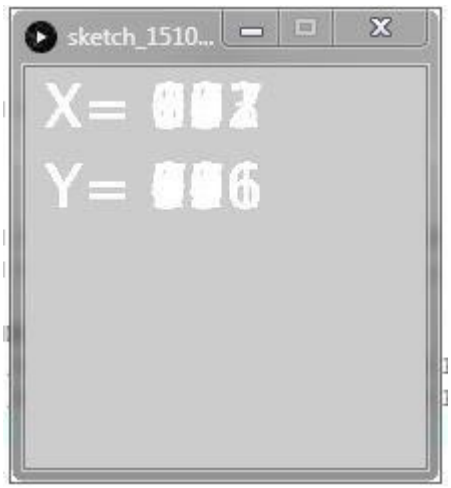
Теоретичні положення

Розглянемо більш ретельно функції, що використовували у попередніх роботах. Функція background () це функція заливки екрану (фон). Використання її у функції draw (), спричинить за собою чимале звуження можливостей, що надаються мовою програмування Processing. Тільки у функції setup() вона і буде виконувати свої прямі обов'язки, в будь-який інший функції вона буде виконувати покадрове зафарбовування екрану.

Щоб краще зрозуміти це, розберемо невеликий приклад. Будемо виводити на екрані значення, які відображають положення курсора щодо осей x і y.

```
int x = 0; //змінна з координатою x
int y = 0; //змінна з координатою y

void setup()
{
 size(200,200); //створюємо вікно 200 на 200 пікселів
}
void draw()
{
 x = mouseX; //записуємо в x координату миша  по осі x
 y = mouseY; // записуємо в y координату миша  по осі y

 textSize(30); //размір тексту
 text("X= "+x, 10, 30); //додаємо текст в координатах 10, 30
 text("Y= "+y, 10, 70); /додаємо текст в координатах 10, 70
}
```

Запускаємо програму і бачимо наступне:

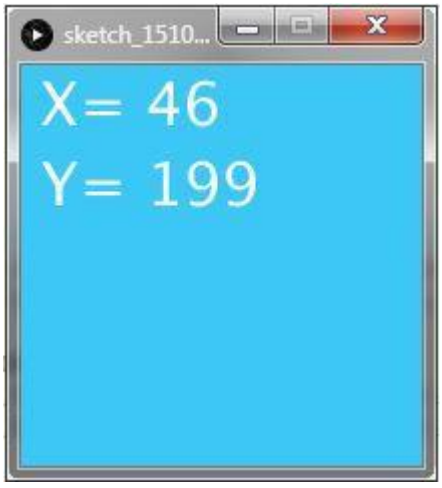


Чому вийшло саме так? Все тому, що в Processing, кожен наступний кадр накладається на попередній, а старий кадр не стирається. Виходить просто накладення кадрів один на одного.

Як вирішити? Додати заливку фону в функцію draw (). Щоб при кожному новому кадрі, попередній пофарбувався.

```
int x = 0;
int y = 0; void setup()
{
 size(200,200);
}
void draw()
{
 background(#3CC7F7);
 x = mouseX;
y = mouseY;

 textSize(30);
text("X= "+x, 10, 30);
text("Y= "+y, 10, 70);
}
```

Получаемо таке:

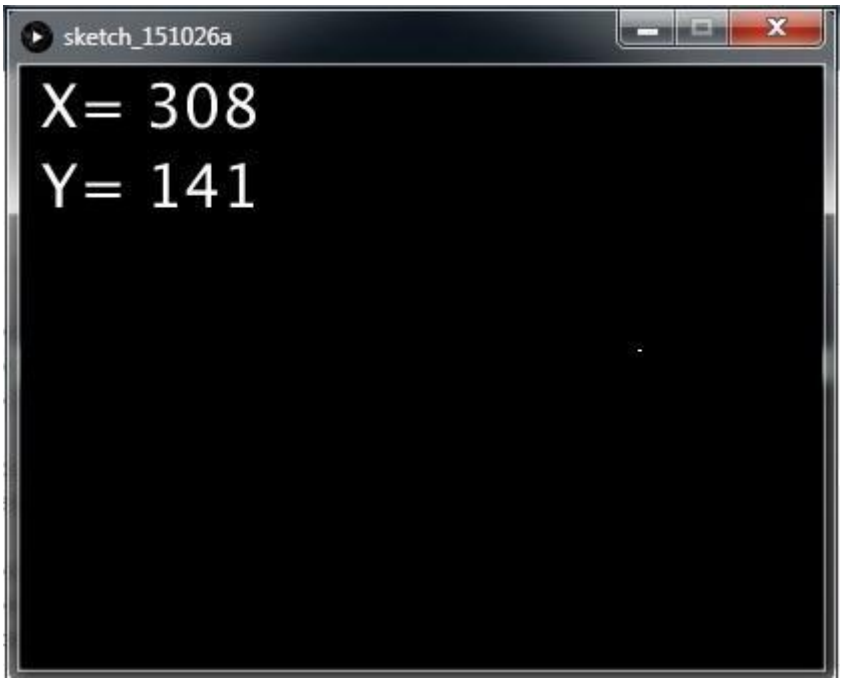


Наступне завдання. Виведемо не тільки значення координат по осях, але а також і графік. Тільки змінювати мишкою будемо тепер координату по

Y. За координатою X змінюватися значення буде автоматично. Давайте запустимо цей код:

```
int xn = 0; //змінна координати x початок
int yn = 0; //змінна координати y початок
int xt = 0; //змінна координати x кінець
int yt = 0; //змінна координати y кінець

void setup()
{
 size (400,300);
}

void draw()
{
 background(0); //заливаємо фон чорним кольором
 yt = mouseY;
 textSize(30);
 text("X= "+xt, 10, 30);
 text("Y= "+yt, 10, 70);
  stroke(255); //виводити график будемо білим кольором
 line(xn,yn,xt,yt); //рисуємо лінію
 yn = yt;
 xn = xt;
 xt = xt+1;
}
```

Запускаємо ти дивимося:



Виникла проблема замість графіка у нас виводиться звичайна точка!

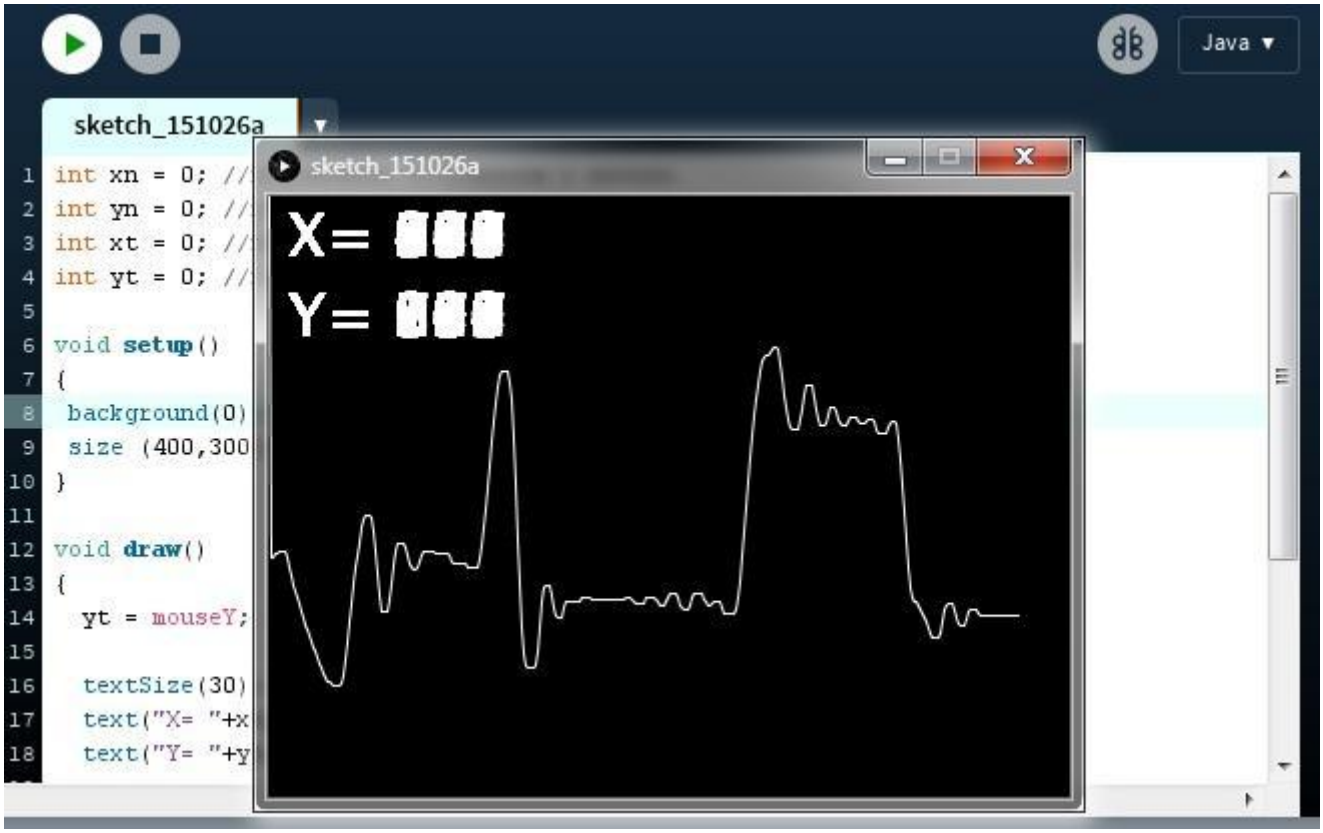
Звичайно ж це через функції background (). Заливка, в будь-який інший функції, буде виконувати покадровое зафарбовування екрану. Таким чином, з кожним новим кадром, у нас буде зафарбовуватися повністю екран, видаляючи попередні лінії графіка.

Щоб це вирішити перенесемо background () в функцію setup():

```
int xn = 0;
int yn = 0;
int xt = 0;
int yt = 0;

void setup()
{
 background(0);
 size (400,300);
}

void draw()
{
 yt = mouseY;
```

```
  textSize(30);
 text("X= "+xt, 10, 30);
 text("Y= "+yt, 10, 70);
  stroke(255);
 line(xn,yn,xt,yt);
  yn = yt;
 xn = xt;
 xt = xt+1;
}
```

Запускаємо.

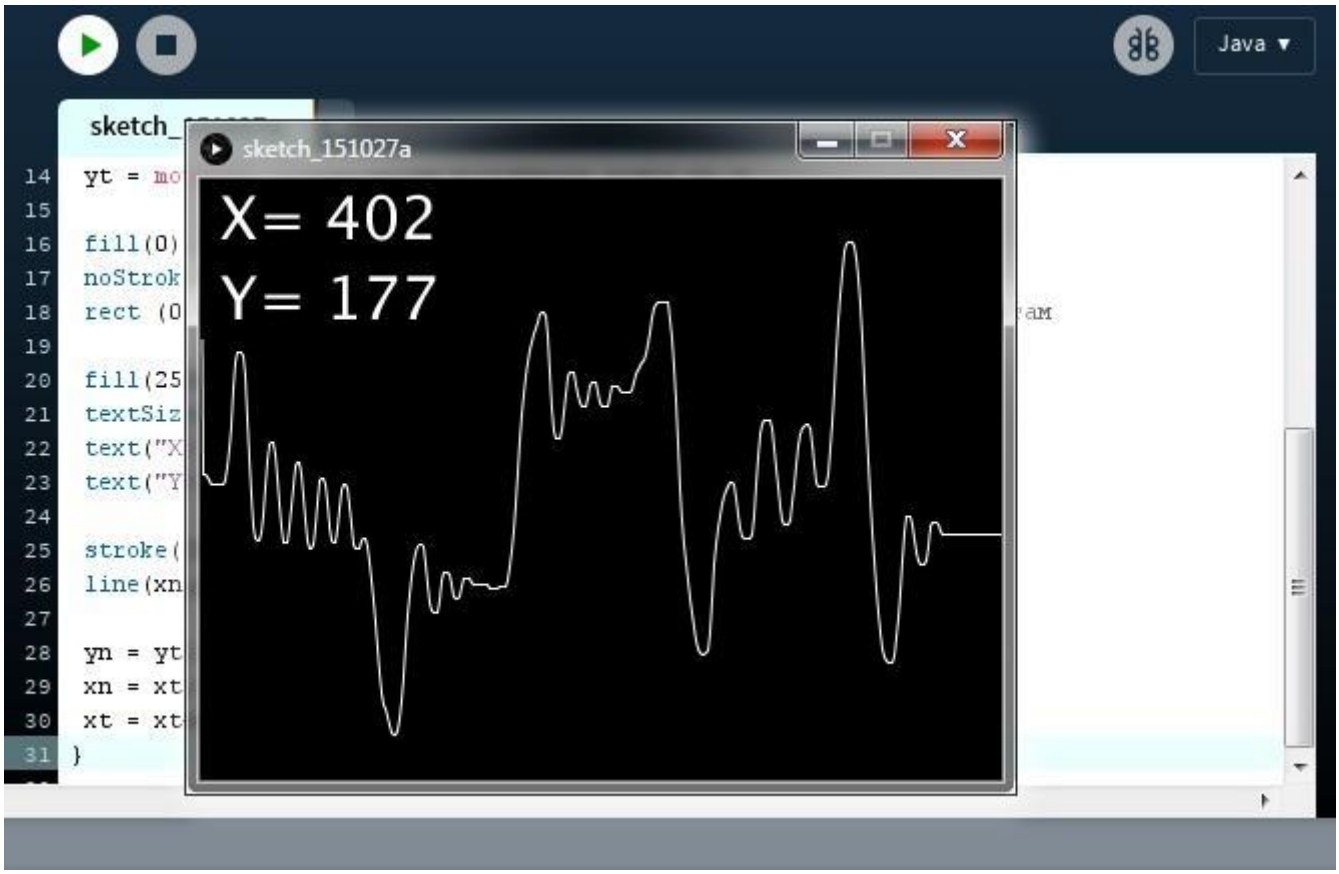


Тепер графік малюється, а ось координати не відображаються. Просто потрібно зафарбовувати кожен кадр тільки в одній області-де відображаються координати. А все інше прати не обов'язково. Так давайте просто додамо прямокутник, і вже на ньому будемо відображати наші значення. Додаємо код:

```
 int xn = 0;
int yn = 0;
int xt = 0;
```

```
int yt = 0;

void setup()
{
 background(0);
 size (400,300);
}

void draw()
{
 yt = mouseY;
  fill(0); //заливка для прямокутника
 noStroke();//малюємо прямокутник без контуру
 rect   (0,0,150,80);   //малюємо   прямокутник   по   визначених
координатах
  fill(255); //колір текста
 textSize(30);
 text("X= "+xt, 10, 30);
 text("Y= "+yt, 10, 70);
  stroke(255);
 line(xn,yn,xt,yt);
  yn = yt;
 xn = xt;
 xt = xt+1;
}
```

Запускаємо програму і бачимо наступне:



Тепер графік і значення відображаються як і повинні. Зверніть увагу, що прямокутник ми додали до додавання тексту. Якби було навпаки, то прямокутник перекривав би весь текст.

На рахунок заливки фігур (fill ()), розмірів тексту (textSize ()), контурів (stroke ()) і т.п, якщо ви хоч раз застосували який-небудь параметр, то протягом всієї програми він залишиться таким же. Для кожного об'єкта необхідно буде задавати свої параметри.

У цій лабораторній роботі зберемо пристрій на платформі Arduino, використовуючи при цьому Потенціометри. Створимо графічну програму, в якій будемо змінювати розміри, положення і поведінку об'єкта на екрані, змінюючи опору резисторів.

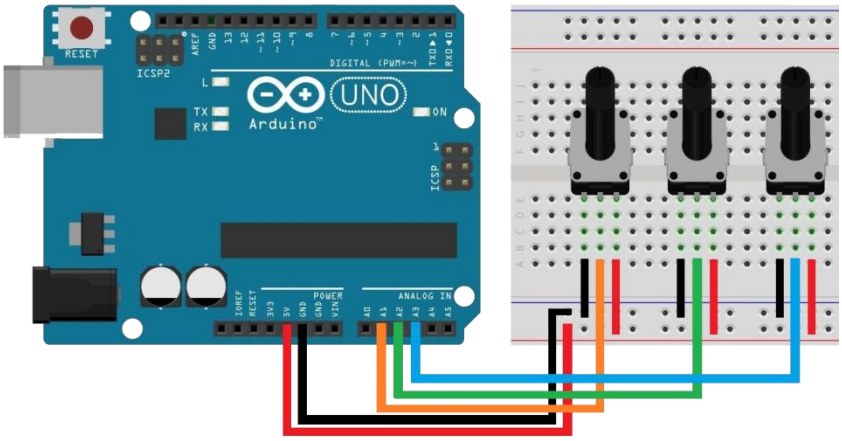
Для роботи нам знадобляться наступні компоненти:

1. Arduino Uno
2. Breadboard
3. З'єднувальні дроти "папа-папа"
4. Перемичка

5. Потенціометри 3шт.

Тепер потрібно зібрати схему пристрою. Вона буде складатися з трьох потенціометрів. Змінювати в роботі ми будемо тільки інтерфейс графічної програми.

Збираємо схему даного пристрою:



І заливаємо цей скетч в Arduino:

```
 int R = A1;
int G = A2;
int B = A3;

void setup()
{
  Serial.begin (9600);
}

void loop()
{
 int val_1 = map(analogRead(R),0,1023,0,255);
 Serial.println(val_1);
 delay(50);
  int val_2 = map(analogRead(G),0,1023,256,511);
 Serial.println(val_2);
 delay(50);
  int val_3 = map(analogRead(B),0,1023,512,767);
```

```
 Serial.println(val_3);
 delay(50);

}
```

Приклад 1

Arduino ми налаштували. Тепер необхідно попрацювати з візуалізацією наших даних, одержуваних з потенціометрів.

Давайте створимо на екрані якийсь об'єкт, що нагадує наш потенціометр. Він буде складатися з кола (ковпачка) і мітки, яка буде вказувати в якому положенні знаходиться потенціометр.

Основою для нашої програми буде служити наступний код:

```
import processing.serial.*;
Serial port;
float ob1 = 0; //перший потенціометр
float ob2 = 0; //другий потенціометр
float ob3 = 0; //третій потенціометр
void setup()
{
 size (300,300);
 port = new Serial(this, "COM5",9600);
 port.bufferUntil('\n');
}
void draw()
{
}
void serialEvent (Serial port)
{
  float val = float (port.readStringUntil ('\n'));
  if ( val >= 0 && val <=255)
 {
 ob1 = map(val,0,255,0,5);
 }
 else if (val >= 256 && val <=511)
 {
```

```
 ob2 = map(val,256,511,0,5);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 }
}
```

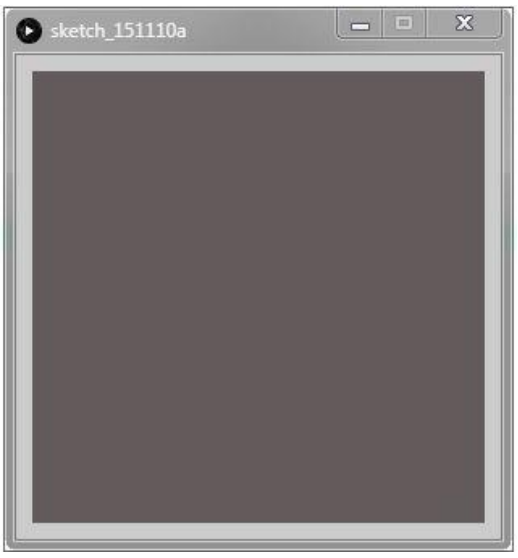
У функції serialEvent () ми зчитуємо значення наших опорів, які надходять на вхід COM порту з Arduino. І переписуємо ці значення, привласнюючи змінним ob1, ob2, ob3, в діапазоні від 0 до 5.

Таким чином, основа для нашої програми готова. Тепер будемо писати в функції draw () ті дії, які будуть виконуватися зі зміною опорів резисторів.

Будемо керувати поведінкою резистора на екрані. Для цього створимо робоче поле, де розташуємо наш потенціометр:

```
void draw()
{/створюємо область розташування потенціометра
 noStroke();
 fill(#625B5B);
 rect(10,10,280,280);
}
```

Для наочності, робимо заливку області відмінну від основного фону програми. Отримуємо наступне:

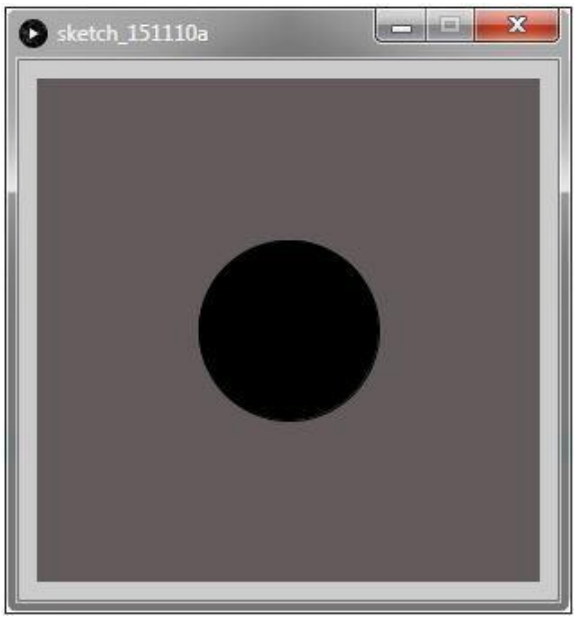


І так, робоча область являє собою прямокутник. З кожним новим кадром, цей прямокутник буде зафарбовувати попередній кадр, щоб виключити накладення і змішування картин.

Тепер в цій області потрібно розташувати модель нашого потенціометра. Для цього додаємо коло:

```
void draw()
{
 noStroke();
 fill(#625B5B);
 rect(10,10,280,280);

//------------------------------/
/*--- Створюємо потенціометр --*/
//------------------------------/
//малюємо ручку потенціометра
 stroke(0);
 fill(0);
 ellipse(150,150,100,100);
 }
```

Додамо окружність чорного кольору в центр вікна, з діаметром 100px. Це і буде основа ковпачка нашого потенціометра. Отримали наступне:



Тепер потрібно додати центральну вісь і мітку положення. Для цього переписуємо код:

```
void draw()
{
 noStroke();
 fill(#625B5B);
 rect(10,10,280,280);
  stroke(0);
 fill(0);
 ellipse(150,150,100,100);
  //малюємо вісь потенціометра
 fill(255);
 ellipse(150,150,30,30);
  //створюємо мітку  на потенціометрі
 strokeWeight(10); //товщина лінії
 stroke(255);
 line(150,150,200,150);
}
```

Отримали ось таку програму:

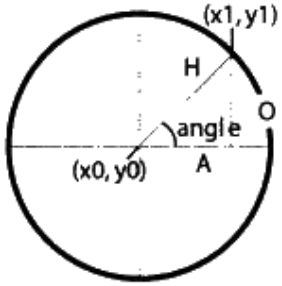
Звичайно це все схематично. Щоб просто налаштувати програму. Коли програма буде працювати, можна буде змінити вигляд потенціометра,

зробити його більш правдоподібним. Але зараз, для розуміння всього процесу, і цього буде достатньо.

Як ви напевно вже зрозуміли, якщо ми будемо змінювати опір потенціометрів, то на екрані нічого відбуватися не буде. Все тому, що ми в функції draw() не використовували одержувані значення з Arduino.

Щоб відображалося, що потенціометр на екрані обертається, потрібно змінювати положення мітки. Точніше її кінцеві координати (x1 і y1), які знаходяться на краю ковпачка.

Як визначити ці самі кінцеві координати? Потрібно згадати тригонометрію. Щоб було більш зрозуміло, подивіться на цей рисунок:



x0 і y0 це початкові координати нашого ковпачка. H - та сама мітка (являє собою лінію). Початкові координати збігаються з центром ковпачка, а кінцеві координати (x1 і y1) розташовуються на окружності. Їх і потрібно знайти.

Щоб знайти координати, потрібно скористатися наступними формулами:

x1 = x0 +(cos (angle) * radius)

y1 = y0 +(sin (angle) * radius)

Де angle – кут, який ми і записуємо в змінні ob1-ob3. Radius – радіус нашого ковпачка (в даному випадку, діаметр ковпачка дорівнює 100px, значить радіус буде дорівнює 50px. x0 і y0 - початкові координати (в даному випадку x0=y0=150).

Тепер давайте додамо ці формули в наш скетч:

```
import processing.serial.*;
Serial port;
```

```
float ob1 = 0;
float ob2 = 0;
float ob3 = 0;

void setup()
{
 size (300,300);
 port = new Serial(this, "COM5",9600);
 port.bufferUntil('\n');
}

void draw()
{
 noStroke();
 fill(#625B5B);
 rect(10,10,280,280);
 stroke(0);
 fill(0);
 ellipse(150,150,100,100);
 fill(255);
 ellipse(150,150,30,30);
  //Обчислюємо кінцеві координати мітки
 float x1 = 150 + (cos(ob1)*50); //координата x1
 float y1 = 150 + (sin(ob1)*50); //координата y1
 strokeWeight(10);
 stroke(255);
 line(150,150,x1,y1);
 }

void serialEvent (Serial port)
{
  float val = float (port.readStringUntil ('\n'));
  if ( val >= 0 && val <=255)
 {
 ob1 = map(val,0,255,0,5);
```

```
 }
 else if (val >= 256 && val <=511)
 {
 ob2 = map(val,256,511,0,5);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 }
}
```

Ось і все. Ми задіяли один потенціометр для зміни поведінки об'єкта в програмі.

Приклад 2

Давайте тепер задіємо всі три потенціометра. Виведемо на екран три змінних резистора, кожен резистор буде відповідати за свій колір, а фон вікна буде відповідати сумі всіх трьох кольорів.

Основу коду ми візьмемо з попереднього прикладу, тільки трохи змінимо. У функції draw () ми створювали наше робоче поле, тут же нам це не знадобиться, тому що будемо заливати постійно все вікно. Скористаємося функцією background (). Аргументи функції будуть всі три складові кольору (color_R, color_G, color_B), що змінюються нашими потенціометрами.

Їх ми зчитуємо з Arduino у функції serialEvent():

```
void serialEvent (Serial port)
{
  float val = float (port.readStringUntil ('\n'));
 if ( val >= 0 && val <=255)
 {
 ob1 = map(val,0,255,0,5);
 color_R = map(val,0,255,0,255);
 }
 else if (val >= 256 && val <=511)
 {
```

```
 ob2 = map(val,256,511,0,5);
 color_G = map(val,256,511,0,255);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 color_B = map(val,215,767,0,255);
 }
}
```

Оскільки ми використовуємо нові змінні, необхідно їх оголосити. Ми це робимо на самому початку програми, перед функцією setup (). Також потрібно записати в функцію draw () функцію заливки екрану, отримуємо наступний код:

```
import processing.serial.*;
Serial port;
float ob1 = 0;
float ob2 = 0;
float ob3 = 0;
float color_R = 0;
float color_G = 0;
float color_B = 0;
void setup()
{
 size (500,200);
 port = new Serial(this, "COM5",9600);
 port.bufferUntil('\n');
}
void draw()
{
 background(color_R,color_G,color_B);
}
void serialEvent (Serial port)
{
  float val = float (port.readStringUntil ('\n'));
  if ( val >= 0 && val <=255)
```

```
 {
 ob1 = map(val,0,255,0,5);
 color_R = map(val,0,255,0,255);
 }
 else if (val >= 256 && val <=511)
 {
 ob2 = map(val,256,511,0,5);
 color_G = map(val,256,511,0,255);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 color_B = map(val,215,767,0,255);
 }
}
```

Якщо ми зараз запустимо програму, то отримаємо таке вікно:

Покрутивши потенціометри, ми побачимо, що заливка екрану змінюється. Тобто ми все зробили правильно. Тепер потрібно додати на екран наші змінні резистори.

Можна просто взяти шматок коду з попереднього прикладу, де створювався один потенціометр і копіювати його для двох інших, змінивши аргументи функцій (кут, координати). Але цей спосіб нераціональний, тому що використовуючи такий метод, ми робимо код більш громіздким. У таких

випадках, найкраще використовувати функції. Тому давайте створимо нову функцію, яка отримавши якісь початкові Дані, сама створить потенціометр. Створюємо функцію, potenz () (створюється за фігурними дужками інших функцій) :

```
void potenz (float x0, float y0, int r,int g, int b, float ug)
{
}
```

Аргументи, які функція буде приймати:

* x0-Початкова координата потенціометра по x

* y0-Початкова координата потенціометра по y

* R-колір потенціометра

* g-Колір потенціометра

* B-колір потенціометра

* UG-значення з Arduino

Тобто, наприклад, якщо ми захочемо намалювати потенціометр з початковими координатами по x і y 200, з заливкою білого кольору (r = 255, g = 255 ,b = 255), і залежить від другого потенціометра, то ми напишемо в функції draw () наступне:

```
potenz(200,200,255,255,255,ob2);
```

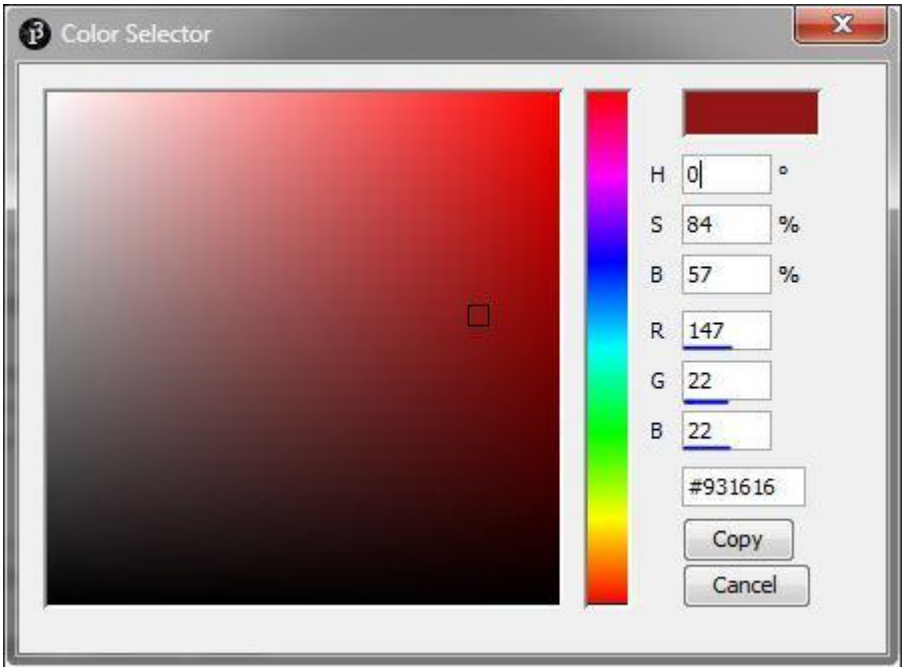
Потрібно описати всі дії з аргументами у функції potenz (). Створюємо ковпачок, мітку і вісь. Тут ви можете самі трохи поекспериментувати. Зробити зовнішній вигляд потенціометра на свій розсуд. Для прикладу, такий варіант:

```
void potenz (float x0, float y0, int r, int g, int b, float
ug)
{
  strokeWeight(1);
 stroke(0);
 fill(#434242);
 ellipse(x0,y0,100,100);
  stroke(r,g,b);
```

```
fill(#747070);
ellipse(x0,y0,80,80);
 fill(r,g,b);
ellipse(x0,y0,30,30);
 float x1 = x0 + (cos(ug)*50);
float y1 = y0 + (sin(ug)*50); 1

strokeWeight(10);
stroke(r,g,b);
line(x0,y0,x1,y1);
}
```

В коді присутні наші аргументи (x0,y0,r,g,b,ug). Ці змінні ми будемо задавати при виклику функції, і від них буде залежати вид і положення об'єкта на екрані.

Функція створення потенціометра готова, тепер потрібно її викликати. Для цього переходимо в функцію draw (). Параметри кольору, для функції potenz () (r,g,b), ви можете взяти з "Color Selector" на панелі, у вкладці " Tools":



Створюємо 3 потенціометра, задаючи для кожного з них свої початкові координати та кольори. Кожен потенціометр залежить від свого (ob1,ob,ob3):

```
potenz(100,100,255,10,39,ob1);
potenz(250,100,6,191,41,ob2);
potenz(400,100,6,65,191,ob3);
```

Кінцевий код програми:

```
import processing.serial.*;
Serial port;
float ob1 = 0;
float ob2 = 0;
float ob3 = 0;
float color_R = 0;
float color_G = 0;
float color_B = 0;
void setup()
{
 size (500,200);
 port = new Serial(this, "COM3",9600);
 port.bufferUntil('\n');
}

void draw()
{
 background(color_R,color_G,color_B);
 potenz(100,100,255,10,39,ob1);
 potenz(250,100,6,191,41,ob2);
 potenz(400,100,6,65,191,ob3);
}

void serialEvent (Serial port)
{
 float val = float (port.readStringUntil ('\n'));
  if ( val >= 0 && val <=255)
 {
```

```
 ob1 = map(val,0,255,0,5);
 color_R = map(val,0,255,0,255);
 }
 else if (val >= 256 && val <=511)
 {
 ob2 = map(val,256,511,0,5);
 color_G = map(val,256,511,0,255);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 color_B = map(val,215,767,0,255);
 }
}

void potenz (float x0, float y0, int r,int g, int b, float ug)
{
 strokeWeight(1);
 stroke(0);
 fill(#434242);
 ellipse(x0,y0,100,100);
 stroke(r,g,b);
 fill(#747070);
 ellipse(x0,y0,80,80);
 fill(r,g,b);
 ellipse(x0,y0,30,30);
 float x1 = x0 + (cos(ug)*50);
 float y1 = y0 + (sin(ug)*50);
 strokeWeight(10);
 stroke(r,g,b);
 line(x0,y0,x1,y1);
 }
```

Приклад 3

Можна запрограмувати об'єкт, який буде змінювати своє розташування на екрані і свої розміри, використовуючи потенціометри. Невеликий приклад, для розуміння самої ідеї.

Код програми:

```
import processing.serial.*;
Serial port;
/* Параметри кубіка */
float x0 = 0; //координата кубіка по x
float y0 = 0; //координата кубіка по y
float d = 10; //довжина кубіка
float h = 10; //висота кубіка
float ob1 = 0;
float ob2 = 0;
float ob3 = 0;
void setup()
{
 size (500,500);
 port = new Serial(this, "COM3",9600);
 port.bufferUntil('\n');
}

void draw()
{
 background(#D38446);
 x0 = x0 + ob1;
 y0 = y0 + ob2;
 d = d + ob3;
 h = h + ob3;
  if (x0 >= width)
 {
 x0 = 0;
 }
 else if (y0 >= height)
 {
 y0 = 0;
```

```
 }
  rect(x0,y0,d,h); //відображаємо кубік
}

void serialEvent (Serial port)
{
 float val = float (port.readStringUntil ('\n'));
  if ( val >= 0 && val <=255)
 {
 ob1 = map(val,0,255,0,5);
 }
 else if (val >= 256 && val <=511)
 {
 ob2 = map(val,256,511,0,5);
 }
 else if (val >= 512 && val <=767)
 {
 ob3 = map(val,512,767,0,5);
 }
}
```

Можна запрограмувати об'єкт, який буде змінювати своє розташування на екрані і свої розміри, використовуючи потенціометри. Невеликий приклад.

Код програми.

```
x0 = x0 + ob1; //зміна координати кубіка по x на значення,
// що отримано з потенціометра 1
 y0 = y0 + ob2; // зміна координати кубіка по y на значення,
//що отримано з потенціометра 2
```

Якщо ми змінюємо опір третього резистора, то ми змінюємо розміри кубика по ширині і висоті.

```
d = d + ob3; //зміна довжини кубіка на значення,
//що отримано з потенціометра 3
 h = h + ob3; // зміна висоти кубіка на значення,
```

```
//що отримано з потенціометра 3
```

Оскільки, кубик може вийти за межі вікна, і ми перестанемо його бачити, в програмі організовано повернення кубика в нульові координати:

```
if (x0 >= width)
 {
 x0 = 0;
 }
 else if (y0 >= height)
 {
 y0 = 0;
 }
```

width та ***height*** - це зарезервовані змінні (їх не треба оголошувати), які позначають ширину і висоту вікна.

Якщо координата x (y) кубика вийшла за межі вікна, то ми координаті x (y) присвоюємо значення 0, і повертаємо кубик в початкове положення.

Виконання

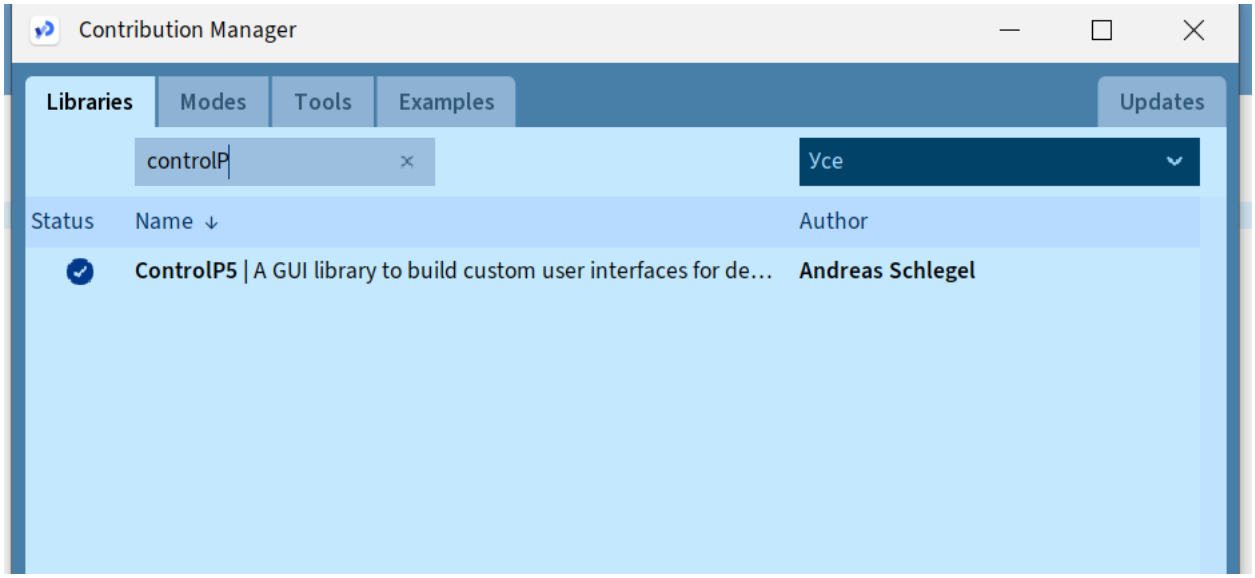
1. Зібрати та запрограмувати наведені експерименти 1, 2, 3.

2. На прикладі 2. Зробіть так, щоб викликаючи функцію potenz (), можна було б також задавати розмір потенціометрів. Тобто потрібно додати в аргумент функції ще одну змінну і змінити трохи саму функцію;

3. На прикладі 3. Переписати програму так, щоб збільшуючи опір першого резистора - кубик переміщався вправо, а при зменшенні опору - вліво. Збільшуючи опір другого-переміщався вниз, при зменшенні-вгору. Змінюючи третій потенціометр, зі збільшенням опору розмір кубика збільшувався, а зі зменшенням - зменшувався. Тобто треба знайти середину потенціометрів. І задати в програмі діапазон, в якому буде рівновага. Наприклад опір змінюється від 0 до 22. Таким чином, діапазон рівноваги потенціометра буде від 10 до 12. Зменшення буде від 0 до 10. Збільшення від 12 до 22. Якщо повернете в одну сторону-відбудеться зменшення опору, в іншу - збільшення.

Лабораторна робота Бібліотека ControlP5

Бібліотека ControlP5 дозволяє додати елементи графічного інтерфейсу у вікно програми-кнопки, перемикачі, списки.



На домашній сторінці містяться приклади до всіх компонентів.

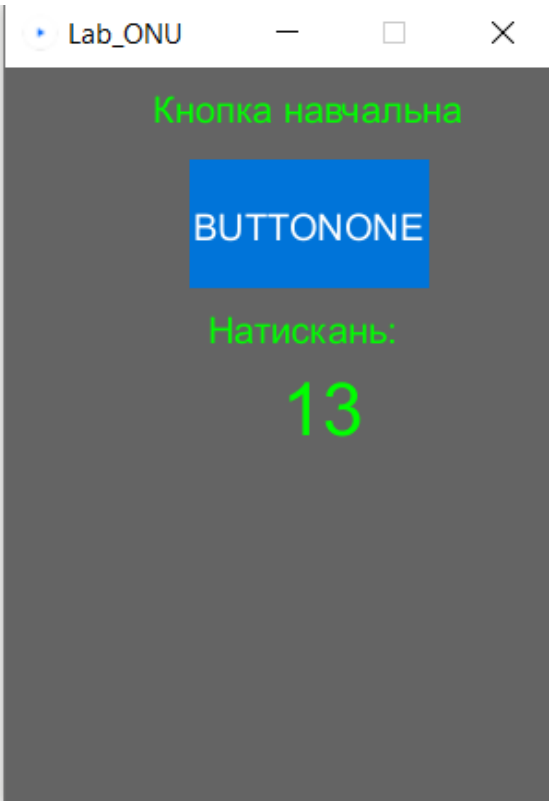
Кнопка

Розглянемо приклад додавання кнопки, яка реагує на клацання.

```
import controlP5.*;
ControlP5 cp5;
PFont font;
int p=0;
void setup() {
  size(300, 400);
```

```
  surface.setTitle("ControlP5_Lab_ONU");
  cp5 = new ControlP5(this);
  font = createFont("calibri light bold", 20);
  cp5.addButton("buttonOne")
    .setPosition(100, 50)
    .setSize(130, 70)
    .setValue(0)
    .setFont(font);
}
void draw() {
  background(100, 100, 100);
  fill(0, 255, 0);
  textFont(font);
  text("Кнопка навчальна", 80, 30);
  text("Натискань: ", 110, 150);
  textFont(font, 40);
  text(p-1, 150, 200);
}
public void buttonOne(int theValue) {
  theValue+=p;
  p++;
  println("Натискань на кнопку - " + theValue);
}
```

У прикладі встановлено позицію та розмір кнопки, а також текст на ній. На додаток встановили власний шрифт замість шрифту за замовчуванням.

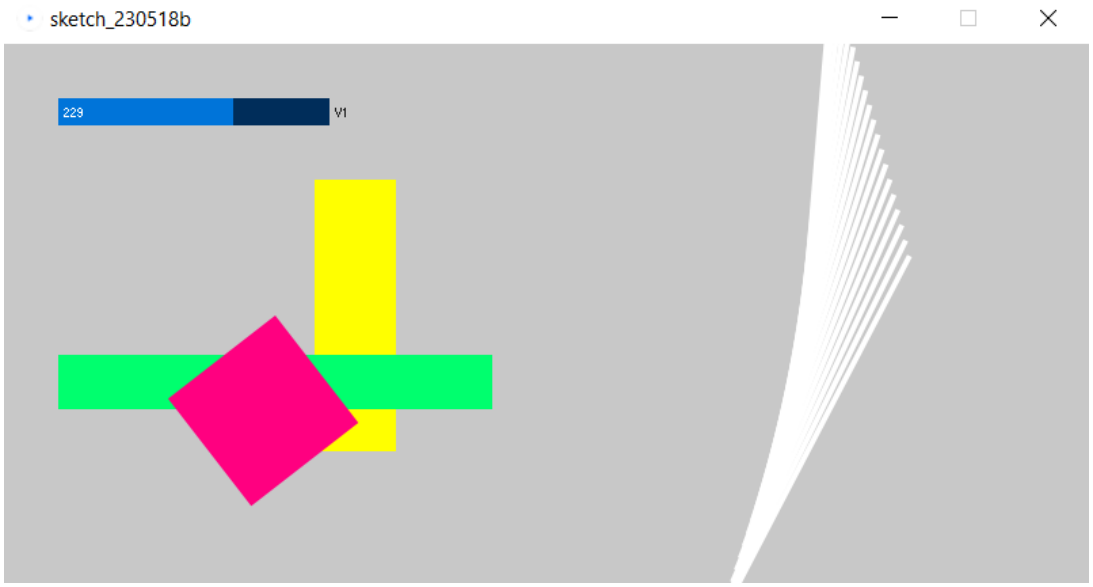


Слайдер

Приклад використання слайдеру для управління процесом

```
import controlP5.*;
  ControlP5 cp5;
  int v1;
  void setup() {
    size(800, 400);
    noStroke();
    cp5 = new ControlP5(this);
    cp5.addSlider("v1") //слайдер
       .setPosition(40, 40)
       .setSize(200, 20)
       .setRange(100, 300)
       .setValue(250)
       .setColorCaptionLabel(color(120,120,120));
  }
  void draw() {
    background(200, 200, 200);
```

```
    pushMatrix();
    pushMatrix();
    fill(255, 255, 0);  // перший прямокутник
    rect(v1, 100, 60, 200);
    fill(0, 255, 110); //другий прямокутник
    rect(40, v1, 320, 40);
    translate(200, 200); // квадрат
    rotate(map(v1, 100, 300, -PI, PI));
    fill(255, 0, 128);
    rect(0, 0, 100, 100);
    popMatrix();
    translate(600, 100); //білі грати
    for (int i=0; i<20; i++) {
      pushMatrix();
      fill(255);
      translate(0, i*10);
      rotate(map(v1+i, 0, 300, -PI, PI));
      rect(-150, 0, 300, 4);
      popMatrix();
    }
    popMatrix();
  }
```



До датково pushMatrix() та popMatrix()

Ставить поточну матрицю перетворення в стек матриць. Розуміння функцій PushMatrix() та PopMatrix () вимагає розуміння концепції стека матриць. Функція PushMatrix() зберігає поточну систему координат у стеку, а функція PopMatrix () відновлює попередню систему координат. PushMatrix() і PopMatrix () використовуються в поєднанні з іншими функціями перетворення і можуть бути вбудовані для управління областю перетворень.

translate()

Задає величину зміщення об'єктів в межах відображуваного вікна. Параметр x визначає переміщення вліво/вправо, параметр y визначає переміщення вгору/вниз, а параметр z визначає переміщення до / від екрана. Використання цієї функції з параметром z вимагає використання P3D як параметра в поєднанні з розміром, як показано у наведеному вище прикладі.

Перетворення є кумулятивними і застосовуються до всього, що відбувається після, а наступні виклики функції накопичують ефект. Наприклад, виклик translate(50, 0), а потім translate(20, 0) - це те саме, що translate(70, 0). Якщо функція translate () викликається всередині функції draw (), перетворення скидається при повторному запуску циклу. Цією функцією можна додатково керувати за допомогою PushMatrix() та PopMatrix().

rotate()

Повертає фігуру на величину, вказану параметром angle. Кути повинні бути вказані в радіанах (значення від 0 до TWO_PI) або перетворені в радіани за допомогою функції radians ().

Об'єкти завжди обертаються навколо їх відносного положення щодо початку координат, а позитивні числа обертають об'єкти за годинниковою стрілкою. Перетворення застосовуються до всього, що відбувається після, і наступні виклики функції накопичують ефект. Наприклад, виклик rotate(HALF_PI), а потім rotate(HALF_PI) - це те саме, що rotate (PI). Усі перетворення скидаються, коли draw() запускається знову.

Технічно функція rotate () множить поточну матрицю перетворення на матрицю обертання. Цією функцією можна додатково керувати за допомогою PushMatrix() та PopMatrix().

Завдання

1. Розглянути приклад роботи програми: приклади/ControlP5/extra/ControlP5frame Розібратися як працює об’єкт Knob. Знайти в прикладах роботу зі списками.
2. Реалізувати функцію передавання даних на стрілковий індекатор за допомогою кнопки, слайдера, списку та Knob.

Лабораторна робота. Бібліотека Net. Створення сервера

Для того щоб створити сервер, нам потрібно вибрати номер порту. Будь-який клієнт, який хоче підключитися до серверу, повинен буде знати цей номер. Номери портів коливаються від 0 до 65 536, і будь-яке число є дійсним, однак порти від 0 до 1024 зазвичай зарезервовані для загальних служб, тому їх краще уникати. Якщо ви не впевнені, чи пошук у Google дасть інформацію про те, чи використовується цей порт іншим застосунком. Для наших цілей ми будем використовувати порт 5204 (це той самий порт, який використовується в довідковій бібліотеці Processing net).

Щоб створити сервер, ми повинні спочатку імпортувати бібліотеку та створити екземпляр серверного об'єкта.

```
Import processing.net.* ;
Server server;
```

Сервер ініціалізується за допомогою конструктора, який приймає два аргументи: "this" і ціле значення для номера порту.

```
server = new Server(this, 5204);
```

Сервер запускається і чекає підключення, як тільки він буде створений. Його можна закрити в будь-який час, викликавши функцію stop( ).

```
server.stop();
```

Ми використовували функцію зворотного виклику (captureEvent( )) для обробки нового кадру відео, доступного з камери. Ми можемо дізнатися, чи підключився новий клієнт до нашого сервера тим же способом, використовуючи функцію зворотного виклику serverEvent ().

serverEvent( ) вимагає двох аргументів: сервера (того, що генерує подію) та клієнта (який підключатися). Ми могли б використовувати цю функцію, наприклад, для отримання IP-адреси підключеного клієнта.

```
// The serverEvent function is called whenever
 // A new client connects
 void serverEvent(Server server, Client client) {
 println( "A new client has connected: " + client.ip());
 }
```

Коли клієнт надсилає повідомлення (після підключення), serverEvent( ) не генерується. Замість цього ми повинні використовувати функцію available (), щоб визначити, чи є нове повідомлення від будь-якого клієнта, доступне для читання. Якщо є, повертається посилання на клієнт, який передає метод, і ми можемо прочитати вміст, використовуючи метод ReadString (). Якщо нічого не доступно, функція поверне значення null, що означає відсутність значення (або не існує клієнтського об'єкта).

```
void draw() {
 // If a client is available, we will find out
 // If there is no client, it will be "null"
```

```
Client someClient = server.available();
// We should only proceed if the client is not null
if (someClient! = null) {
println( "Client says: " + SomeClient.readString());
}
}
```

Функція ReadString () корисна в додатках, де текстова інформація передається по мережі. Якщо дані повинні оброблятися по-іншому, наприклад, як число, можуть бути викликані інші методи read ().

Сервер також може надсилати повідомлення клієнтам, і це робиться за допомогою методу write( ).

```
server.write( "Great, thanks for the message!\n ");
```

Залежно від того, що ви робите, часто є гарною ідеєю надіслати символ нового рядка в кінці вашого повідомлення. Керуючою послідовністю для додавання символу нового рядка в рядок є '\n'.

Події сервера відбуваються тільки при підключенні нового клієнта.

Об'єднавши все вищесказане, ми можемо написати простий чат-сервер. Сервер відповідає на будь-яке отримане ним повідомлення повторенням повідомлення.

```
import processing.net.*;
// Declare a server
Server server;

// Used to indicate a new message has arrived
float newMessageColor = 255;
String incomingMessage = "";

void setup() {
  size(400,200);
    // Create the Server on port 5204
  server = new Server(this, 5204);
}
```

```
     void draw() {
       background(newMessageColor);
        // newMessageColor fades to white over time
       newMessageColor = constrain(newMessageColor + 0.3,0,255);
       textAlign(CENTER);
       fill(0, 255, 0);

// The most recent incoming message is displayed in the window.
       text(incomingMessage,width/2,height/2);
       // If a client is available, we will find out
       // If there is no client, it will be"null"
       Client client = server.available();
       // We should only proceed if the client is not null
       if (client != null) {
         // Receive the message
         // The message is read using readString().
         incomingMessage = client.readString();
// The trim() function is used to remove the extra line break
// that comes in with the message.
         incomingMessage = incomingMessage.trim();
         // Print to Processing message window
         println("Client says: " + incomingMessage);

// Write message back out (note this goes to ALL clients)
server.write("How  does  "  +  incomingMessage  +  "  make  you
feel?\n");
     // A reply is sent using write().
     // Reset newMessageColor to black
         newMessageColor = 0;
       }
     }
//  The  serverEvent  function  is  called  whenever  a  new  client
//connects.
     void serverEvent(Server server, Client client) {
```

```
        incomingMessage = "A new client has connected: " +
client.ip();
        println(incomingMessage);
        // Reset newMessageColor to black
        newMessageColor = 0;
      }
```

Після запуску сервера ми можемо створити клієнт, який підключається до сервера. Зрештою, ми розглянемо приклад, коли ми пишемо як сервер, так і клієнт у процесі обробки. Однак, просто щоб продемонструвати, що

сервер працює, ми можемо підключитися до нього за допомогою будь-якої клієнтської програми telnet. Telnet - це стандартний протокол для віддалених підключень, і всі комп'ютери, як правило, оснащені вбудованими можливостями telnet. На Комп'ютері Mac запустити термінал, у Windows перейдіть до командного рядка.

Оскільки ми підключаємося до сервера з того самого комп'ютера, на якому запущений сервер, адреса, на який ми підключаємося по telnet, - localhost, що означає локальний комп'ютер, порт 5204. Ми також могли б використовувати адресу 127.0.0.1 .

Це спеціальна адреса, зарезервована для того, щоб програми на комп'ютері спілкувалися один з одним локально (тобто на одному комп'ютері) і є еквівалентом localhost. Якби ми підключалися з іншого комп'ютера, нам потрібно було б знати мережеву IP-адресу машини, на якій працює сервер.

Як увімкнути telnet.exe у Windows 10

telnet.exe-це штатний телнет-клієнт в операційних системах сімейства Windows. Він дозволяє дистанційно керувати різними мережевими пристроями (комутаторами, роутерами, серверами і т.п.) по протоколу Телнет. Присутній у всіх версіях операційних систем Microsoft. В Windows 10 за замовчуванням деактивовано і щоб запустити telnet.exe потрібно додатково активувати його в компонентах.

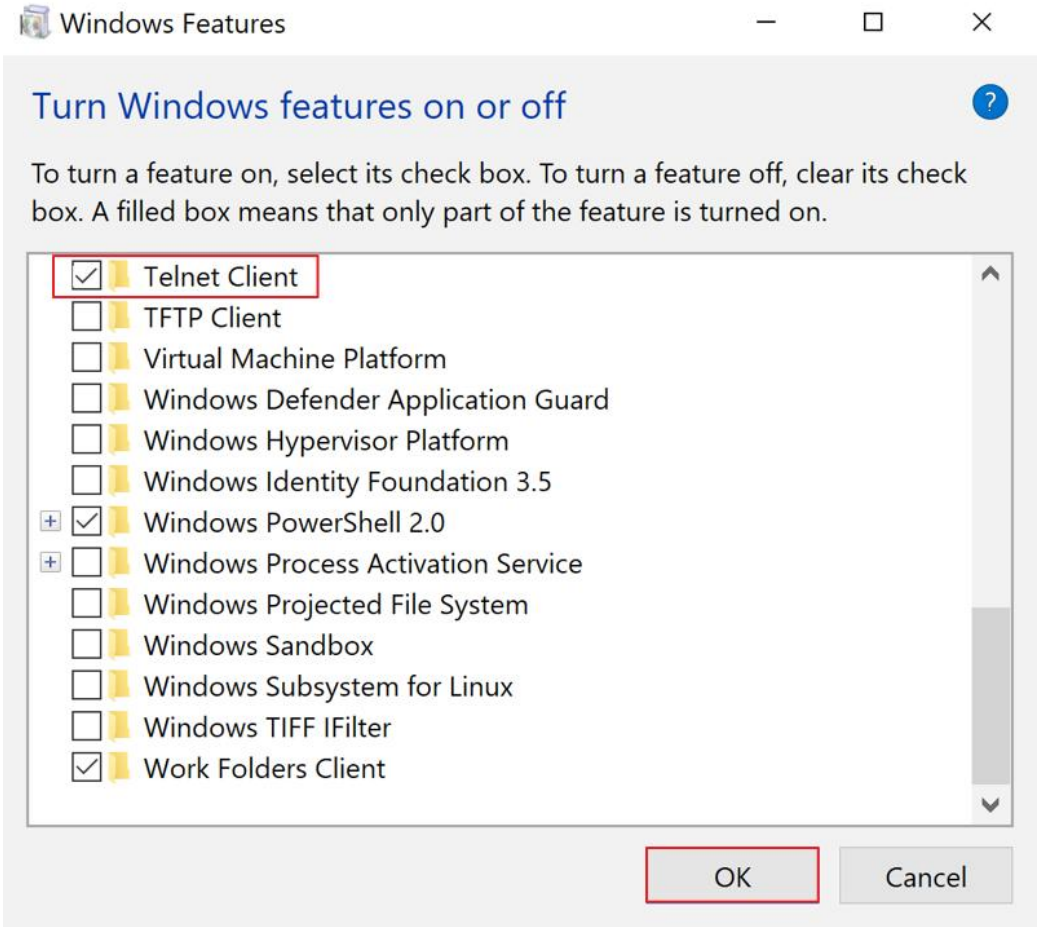
Включаємо telnet в Віндовс:

Послідовність дій наступна:

1. Заходимо в Панель управління Віндовс 10 і знаходимо розділ «Програми та компоненти». Потрапивши в нього, в меню праворуч знаходимо пункт " включення або відключення компонентів Windows»:

Відкриється вікно "Компоненти Windows" в якому треба знайти і поставити галку на клієнт Telnet.

Тиснемо на " ОК " і чекаємо виконання змін:



Як запустити Telnet в Віндовс:

Натискаємо комбінацію клавіш Win + R щоб відкрити вікно “Виконати”:

У рядок “Відкрити” вводимо команду cmd і натискаємо на »ОК".

Так ми запустимо командний рядок Віндовс.

Потім вводимо команду telnet і натискаємо клавішу «Enter». Телнет-клієнт Windows 10 запущений і можна ним користуватися.

Для відкриття відповідного порта команда

```
o localhost 5204
```

Завдання

Запустити програму сервера, та перевірити зв'язок за допомогою терміналу або Telnet

Лабораторна робота Бібліотека Net. Broadcasting.

У лабораторній роботі розглянемо написання сервера широкомовної трансляція, який передає числові дані клієнтам.

Чим це корисно? Що, якби ви хотіли безперервно транслювати температуру за межами вашого будинку, котирування акцій або кількість об’єктів, що зафіксовані системою контролю доступу у будинку?

Щоб продемонструвати фреймворк для такої програми, ми напишемо сервер, який транслює число від 0 до 255 (ми можемо надсилати лише один байт за раз). Потім ми розглянемо клієнтів, які витягують дані та інтерпретують їх по-своєму.

Ось сервер, який випадковим чином збільшує число і транслює його.

```
import processing.net.*;
 // Declare a server
 Server server;
 int data = 0;
 void setup() {
 size(200,200);
 // Create the Server on port 5204
 server= new Server(this, 5204);
  }
 void draw() {
 background(255);
 // Display data
 textAlign(CENTER);
 textSize(64);
 fill(0);
 text(data,width/2,height/2);
 // Broadcast data
 server.write(data);
 // Arbitrarily changing the value of data randomly
 data = (data + int(random( – 2,6))) % 256;
 delay(250);
 }
 // The serverEvent function is called whenever a new client
//connects.
```

```
void serverEvent(Server server, Client client) {
println( " A new client has connected: " + client.ip());
}
```

Далі ми напишемо 2 клієнта, які отримує число із сервера і використовують його для заповнення змінної. Приклад написаний з припущенням, що сервер і клієнт працюють на одній машині (ви можете відкрити обидва приклади та запустити їх разом у процесі обробки), але в реальному сценарії це, швидше за все, буде не так. Якщо ви вирішите запускати сервер і клієнти на різних комп'ютерах, машини повинні бути підключені до локальної мережі (через маршрутизатор або концентратор, ethernet або Wi-Fi) або до Інтернету. IP-адресу можна знайти в мережевих налаштуваннях вашого комп'ютера.

Клієнт 1. Виводить зміну фону відповідно до числа, що надсилає сервер.

```
// Import the net libraries
 import processing.net.*;
 // Declare a client
 Client client;
 // The data we will read from the server
 int data;
 void setup() {
 size(200,200);
 // Create the Client
 client = new Client(this, "127.0.0.1", 5204);
 }
void clientEvent(Client client){
data = client.read(); // Read data
}
 void draw() {
```

```
 background(data);
 }
```

Клієнт 2. Керує кутом повороту квадрату відповідно до числа, що надсилає сервер.

```
// Import the net libraries
 import processing.net.*;
 // Declare a client
 Client client;
 // The data we will read from the server
 int data;
 void setup() {
 size(200,200);
// Create the Client
 client= new Client(this, " 127.0.0.1 " , 5204);
 }
 void clientEvent(Client client){
data = client.read(); // Read data
}
 void draw() {
 background(255);
 stroke(0);
 fill(175);
 translate(width/2,height/2);

 float theta = map(data, 0, 255, 0, TWO_PI);
 rotate(theta);
 rectMode(CENTER);
 rect(0,0,64,64);
```

```
}
```

Завдання

1. Побудувати ще два клієнта (третій та четвертий). Третій – виводити данні у вигляді стрілкового індикатора, четвертий – у вигляді графіку у декартових координатах.
2. Переробити перший клієнт на вивід кольорового фону (background(dataR,dataG,dataB);)

Лабораторна робота Бібліотека Net. Багатокористувацька взаємодія.

Приклад трансляції у попередньої лабораторної роботі демонструє однобічний зв'язок, коли сервер транслює повідомлення, і багато клієнтів отримують це повідомлення. Однак широкомовна модель не дозволяє клієнту розвернутися і відправити відповідь назад на сервер. У цієї роботі

розглянемо, як створити скетч, який передбачає взаємодію між кількома клієнтами, що підтримується сервером.

Давайте розглянемо, як працює чат-кімната. Наприклад, п'ять клієнтів (ви та четверо друзів) підключаються до сервера. Один клієнт набирає повідомлення: "всім привіт! Це повідомлення надсилається на сервер, який передає його назад усім п'яти клієнтам.

Більшість багатокористувацьких додатків функціонують аналогічним чином. Наприклад, у багатокористувацькій онлайн-грі клієнти, швидше за все, відправляють інформацію, пов'язану з їх місцезнаходженням і діями, на сервер, який передає ці дані назад усім іншим клієнтам, які грають у гру.

Багатокористувацька програма може бути розроблена в процесі обробки за допомогою мережевої бібліотеки net.

Щоб продемонструвати це, створимо мережеву загальну дошку. Коли клієнт переміщує курсор миші по екрану, ескіз відправить координати X, Y на сервер, який передасть їх назад будь-яким іншим підключеним клієнтам. Кожен підключений зможе бачити дії з малювання всіх інших.

На додаток до вивчення того, як взаємодіяти між кількома клієнтами, у цьому прикладі буде розглянуто, як надіслати кілька значень. Як клієнт може надіслати два значення (координати X та Y) та повідомити серверу , яке з них є яким?

Перший крок до рішення включає розробку протоколу для спілкування між клієнтами. У якому форматі надсилається інформація та як ця інформація приймається та інтерпретується?

Припустимо, клієнт хоче надіслати розташування миші: mouseX 150 та mouseY 125. Нам потрібно відформатувати цю інформацію у вигляді рядка таким чином, щоб її було зручно розшифровувати. Одна з можливостей полягає в наступному:

"Перше число перед комою - це розташування X, друге число після коми - Y. Наші дані закінчуються там, де з'являється Зірочка (*). ”

У коді це виглядало б наступним чином:

```
String dataToSend = " 100,125*";
```

або, в більш загальному плані:

```
String dataToSend = mouseX+", " + mouseY+"*";
```

Тут ми розробили протокол для надсилання та отримання даних. Цілі значення для mouseX та mouseY кодуються як рядок під час надсилання (число, за яким слідує кома, за яким слідує число, за яким слідує Зірочка). Вони повинні бути розшифровані при отриманні. Слід також зазначити, що більшість прикладів зазвичай використовують символ нового рядка або повернення каретки для позначення кінця повідомлення. Ми використовуємо тут зірочку з двох причин:

1) Зірочка добре видно при відображенні, тоді як символ нового рядка - ні (особливо в контексті книги) та

2) Використання зірочки демонструє, що ви можете розробити та реалізувати будь-який протокол обміну повідомленнями, який ви виберете, за умови, що він відповідає клієнтський та серверний частинам.

Довідка

Дані, що передаються по мережі, передаються у вигляді послідовного списку окремих байтів. Відзначимо, що байт - це 8-розрядне число, тобто число, що складається з восьми 0 і 1 або значення від 0 до 255.

Припустимо, що ми хочемо надіслати число 42. У нас є два варіанти:

```
client.write (42); / / надсилання байта 42
```

У наведеному вище рядку ми фактично надсилаємо фактичний байт 42.

```
client.write ("42"); / / надсилання рядка " 42 "
```

У рядку вище ми відправляємо рядок. Рядок складається з двох символів, ' 4 ' і ' 2 ' .

Ми відправляємо два байти. Ці байти визначаються за допомогою коду ASCII, стандартизованого засобу кодування символів. Символ ' A ' це байт 65, символ ' B ' 66 і так далі. Символ " 4 "- це байт 52, а " 2 " - 50.

Коли ми читаємо дані, нам вирішувати, чи хочемо ми інтерпретувати вхідні байти як буквені числові значення або як ASCII-коди символів. Ми досягаємо цього, вибираючи відповідну функцію read().

```
int val client.read (); / / збігається з client.write(42);
String s = client.ReadString (); / / те саме, що і client.write
//( " 42 ");
int num int (s); / / перетворює прочитаний рядок у число
```

Тепер ми готові створити сервер для прийому повідомлень від клієнта. Завданням клієнта буде відформатувати ці повідомлення відповідно до нашого протоколу. Робота сервера залишається простою: 1) отримувати дані та 2) ретранслюйте дані.

Крок 1. Отримання даних.

```
Client client = server.available();
if (client! = null) {
incomingMessage = client. readStringUntil( ' * ' );
}
```

Новим у цьому прикладі є функція readStringUntil(). Функція readStringUntil () приймає один аргумент - зірочку. Аргумент - зіроччка використовується для позначення кінця вхідних даних. Ми просто слідуємо протоколу, встановленому при відправці. Ми можемо це зробити, тому що розробляємо як сервер, так і клієнт.

Як тільки ці дані будуть прочитані, ми будемо готові додати:

Крок 2. Ретрансляція даних назад клієнтам.

```
Client client = server.available();
 if (client ! = null) {
 incomingMessage = client.readStringUntil( ' * ' );
 server.write(incomingMessage);
 }
```

Код серверу.

Повідомлення відображається на екрані при підключенні нових клієнтів, а також при отриманні даних сервером.

```
// Import the net libraries
import processing.net.*;
// Declare a server
Server server;
String incomingMessage = "";

void setup() {
  size(400, 200);
  // Create the Server on port 5204
  server = new Server(this, 5204);
}

void draw() {
  background(255);
  // Display rectangle with new message color
  fill(0);
  textSize(20);
  textAlign(CENTER);
  text(incomingMessage, width/2, height/2);
  // If a client is available, we will find out
  // If there is no client, it will be"null"
  Client client = server.available();
  // We should only proceed if the client is not null
  if (client != null) {
    // Receive the message
    incomingMessage = client.readStringUntil('*');
```

```
    // Print to Processing message window
    println( "Client says:" + incomingMessage);
    // Write message back out (note this goes to ALL clients)
    server.write(incomingMessage);
  }
}

// The serverEvent function is called whenever a new client
connects.
void serverEvent(Server server, Client client) {
  incomingMessage = "A new client has connected:" + client.ip();
  println(incomingMessage);
}
```

Робота клієнта складається з трьох частин:

1. Надішліть координати mouseX та mouseY на сервер.

2. Витягніть повідомлення з сервера.

3. Відобразіть крапки у вікні на основі повідомлень сервера.

Всі повідомлення, отримані від одного клієнта, негайно передаються назад всім клієнтам за допомогою функції write ().

Для кроку 1 нам потрібно дотримуватися протоколу, який ми встановили для надсилання:

mouseX кома mouseY зірочка

```
String out = mouseX +" , " + mouseY+ " * " ;
client.write(out);
```

Залишається питання: коли настане правильний час для надсилання цієї інформації? Ми могли б вставити ці два рядки коду в основний цикл draw (), відправляючи координати миші в кожному кадрі. Однак у випадку клієнта нам потрібно відправляти координати тільки тоді, коли користувач водить мишею по вікну.

Функція mouseDragged() - це функція обробки подій, подібна до функції mousePressed(). Замість щоб викликати, коли користувач клацає мишею, він викликається щоразу, коли відбувається подія перетягування, тобто миша кнопка натиснута, і миша переміщається. Зверніть увагу, що функція викликається безперервно, коли користувач переміщує курсор миша.

```
void mouseDragged() {
 String out = mouseX +" , " + mouseY+ " * " ;
 // Send the String to the server
 client.write(out);
 // Print a message indicating we have sent data
 println( " Sending: " + out);
 }
```

Крок 2, отримання повідомлень із сервера, працює майже так само, як у прикладах клієнта мовлення попередньої роботи. Єдиною відмінністю є використання функції readStringUntil(), яка відповідає протоколу "number-comma-number-asterisk ".

```
if (client.available() > 0) {
 // Read message as a String, all messages end with an asterisk
 String in = client.readStringUntil( ' * ' );
 // Print message received
 println( " Receiving: "+ in);
 }
```

Спочатку рядок розбивається на масив рядків, використовуючи кому (або зірочку) як роздільник.

```
String[] splitUp = split(in, " ,* " );
```

Потім масив рядків перетворюється на масив цілих чисел (довжина: 2).

```
int[] vals = int(splitUp);
```

І ці цілі числа використовуються для відображення еліпса.

```
fill(255,100);
```

```
 noStroke();
 ellipse(vals[0],vals[1],16,16);
```

Ось повний скетч клієнта:

```
import processing.net.*;
// Declare a client
Client client;


void setup() {
  size(200, 200);
  // Create the Client
  client = new Client(this, "127.0.0.1", 5204);
  background(255);
}


void draw() {
   String in = client.readStringUntil('*');
  if (in != null) {
    // Print message received
    println( "Receiving:" + in);
    // The client reads messages from the Server and parses them
with splitTokens() according to our protocol.
    int[] vals = int(splitTokens(in, ",*"));
    // Render an ellipse based on those values
    fill(0, 100);
    noStroke();
    ellipse(vals[0], vals[1], 16, 16);
  }
}
// If there is information available to read from the Server
```

```
void clientEvent(Client client) {
  // Read message as a String, all messages end with a newline
character
  }


// Send data whenever the user drags the mouse
void mouseDragged() {
  // Put the String together with our protocol: mouseX comma
mouseY newline
  String out = mouseX + "," + mouseY + "*" ;
  fill(0, 100);
  noStroke();
  ellipse(mouseX, mouseY, 16, 16);  // A message is sent
whenever the mouse is dragged. Note that a client will receive
its own messages! Nothing is drawn here!
  client.write(out);
  // Print a message indicating we have sent data
  println("Sending: " + out);
}
```

Завдання

1. Запустити сервер и два-три клієнта и перевірити роботу системи.
2. Модифікувати протокол передачі та код програми таким чином, щоб кожен клієнт малював окремим кольором на інших.