# PROCESSING

## ОСНОВИ ПРОГРАМУВАННЯ

# PROCESSING

## ОСНОВИ ПРОГРАМУВАННЯ

Навчання програмуванню може здатися досить складним, як для дітей, так і для дорослих, які тільки починають опановувати це мистецтво. На щастя, існує багато мов і середовищ програмування для будь-якого рівня майстерності та досвіду.

Якщо вам потрібен інструмент, який чудово підходить для:

- дітей та дорослих, які не мають досвіду програмування;
- того, щоб показати дітям, наскільки цікавим може бути програмування;
- подання основних понять та концепцій алгоритмізації та програмування;
- розвитку логічного та алгоритмічного мислення;
- створення візуальних творів мистецтва на екрані комп'ютера;
- розвитку навичок 21 століття*(The Future of Jobs. Report 2020)*:
  - аналітичного мислення і інноваційності;
  - постійного активного навчання;
  - критичного мислення і аналізу;
  - креативності, оригінальності та ініціативності;
  - використання технологій у розробці та програмуванні;
  - вміння розв'язувати проблеми та формувати ідеї

- ви знайшли саме його!

## Що таке Processing?

Мова програмування - це набір спеціальних інструкцій та правил їх запису у програмах для обчислювальної машини, які є записом алгоритмів у формі, зрозумілій для виконання комп'ютером.

Десятки років вивчення програмування передбачало введення коду, який маніпулював даними за допомогою суто математичних операцій, типовими завданнями були знаходження найбільшого спільного дільника, коренів квадратного рівняння та знаходження чогось за допомогою методу Монте-Карло (це не так страшно, як може задтися на перший погляд). Саме тому у більшості людей, не знайомих з азами алгоритмізації та роботи з комп'ютером, вичення мови програмування завжди викликає певні побоювання: "Мені доведеться вивчити математику та десятки незрозумілих англійських слів у самих неймовірних комбінаціях, перш ніж я зможу зробити щось наочне?"

Програмування - це є як написання коду, так і спосіб думати, як зробити щось нове та унікальне, а також обійти деякі обмеження існуючих програмних засобів.

Ми познайомимо вас з мовою програмування, яка була створена ще в 2002 році для роботи графічними об'єктами - геометричними фігурами та зображеннями: вона називається Processing. Цей посібник допоможе

зрозуміти, що таке Processing та чому ви повинні навчитися програмуванню у ньому.

Головна ідея при вивченні Processing полягає в тому, щоб починаючи з першого рядка програми розпочати створення візуальних об’єктів, якими можна оперувати, змінюючи параметри інструкцій з яких складається програма. Разом з тим, вивчаються ті самі речі, як і при вивченні інших мов програмування - ви дізнаєтесь про змінні, функції, об'єкти та масиви, але все це робиться за допомогою «візуального» дослідження властивостей створюваних графічних зображень.

Processing - це повністю функціональна мова програмування, заснована на Java, де першою вправою є вже не "Hello World", а скоріше "Намалюйте смайлик" на екрані комп’ютера. Разом з тим, Processing - це і професійний інструмент, який використовується для сотень проектів високого класу в широкому діапазоні областей, від мультимедійних установок до візуалізації інформації, це не іграшка чи «навчальна» мова, як Скретч, незважаючи на своє коріння як навчальний засіб.

Найкраще, що простота Processing захоплює дітей програмуванням! Його можна використовувати для створення інтерактивних програм, ігор та анімації, що допомагає при навчанні та опануванні концепцій «дорослого» програмування.

Processing побудовано на мові програмування Java, самне тому вам не доведеться робити надто великих зусиль при переході на інші мови програмування, таких як C/C++. Мова програмування мікроконтролерів Arduino теж має корені у Processing.

Processing постачається з невеликим, але досить ефективним середовищем розробки (IDE), чудовою документацією, великою бібліотекою розширень та значним набором прикладів та демонстраційних програм, він є безкоштовним, з відкритим кодом і добре задокументовим - все це робить його дуже доступним.

## Що «під капотом»?

Як мова програмування, Processing виступає лише як "шар" поверх Java. Весь код Processing спочатку перекладається на код Java. Це означає, що ви можете писати код Java та імпортувати бібліотеки Java у свій код Processing в (або за межами) Processing IDE. Методично це допомагає Processing служити мовою програмування яка має «безшовний» перехід до Java та інши повнофункціональни мов, таких як C/C++. Ви можете розпочати програмування за допомогою Processing, фактично опанувавши Java у звичному середовищі, а потім перейти до потужних інструментів.

**Processing - Java.**

Це насправді просто Java, з дещо простішим синтаксисом, зі спрощеним маніпулюванням графікою у поєднанні з простим у використанні IDE. Коли ви натискаєте «Запустити», Processing запускає ваші файли .pde через препроцесор, який перетворює їх в один гігантський клас Java. Ваші класи - це всі внутрішні класи вашого основного класу, який розширює PApplet. Всі ваші користувацькі об'єкти можуть отримати доступ до «глобальних» змінних та функцій, визначених у вашому основному класі, а також до всіх методів Processing API. Це означає, що ви можете використовувати багато класів зі стандартної бібліотеки Java. Разом з тим буде гарною ідеєю спробувати відкинути свої уявлення про Java під час першого навчання Processing.

## Якщо ви прагнете більшого

Існує хибне сприйняття Processing як надто простої мови програмування, однак Processing - це також інструмент, який полегшує життя навіть досвідченим розробникам, він має є набагато більше, адже спільнота Processing надає бібліотеки, які розширюють функціональні можливості середовища на комп'ютерне бачення, роботу з аудіопристроями та різними типами інтерфейсу, таких як Kinect або Leap Motion. Існують бібліотеки для експорту PDF-файлів, роботи з вебкамерами, створення 3D-зображень, анімації, надсилання текстових повідомлень SMS, включаючи дані про погоду, створення типографіки та багато іншого.

Окрім цих можливостей, протягом багатьох років люди розробляли інструменти, які можуть зробити ваші навички Processing корисними у багатьох контекстах. Кілька прикладів:

- веб/браузер: Processing.js - це бібліотека JavaScript, яка дозволяє вам запускати код Processing в браузері. p5.js - це бібліотека за допомогою якої ви можете конвертувати свої ескізи для інтеграції з вашими вебсайтами.
- мобільні пристрої: Ви можете розробляти програми для Android за допомогою Processing, використовуючи IDE у режимі «Android".
- електроніка, IoT... : мова програмування Wiring та середовище Arduino дуже схожі на Processing.

Ви також можете імпортувати функціональність Processing у свої проекти Java. Це дозволяє використовувати швидкість та простоту Processing для мультимедіа тощо у контексті складних застосунків, які потребують більш повнофункціональної мови програмування.

## Продовжуйте своє навчання...

Варто пам’ятати, що програмісти не можуть знати усіх існуючих іструкцій мови програмування - вони їх шукають у посібниках, з’ясовують, як вони працюють, потім перевіряють їх роботу. Використовуйте зручний довідник, який є у середовищі програмування та на сайті Processing.org ( References) - він дасть відповіді на багато питань під час програмування.

Найкращими книгами, з якими варто ознайомитись є «Processing: Підручник з програмування для візуальних дизайнерів та художників» Кейсі Ріса та Бена Фрая, авторів Processing. Це гарна книга, ретельно відредагована та повна джерел для натхнення.

Вам сподобався Даніель Шиффман із навчальних відеороликів Hello Processing? Він є автором фантастичної книги під назвою Learning Processing: A Beginner's Guide to Programming Images. У цій книзі доступно пояснюється процес програмування, кожна глава включає творчі вправи.

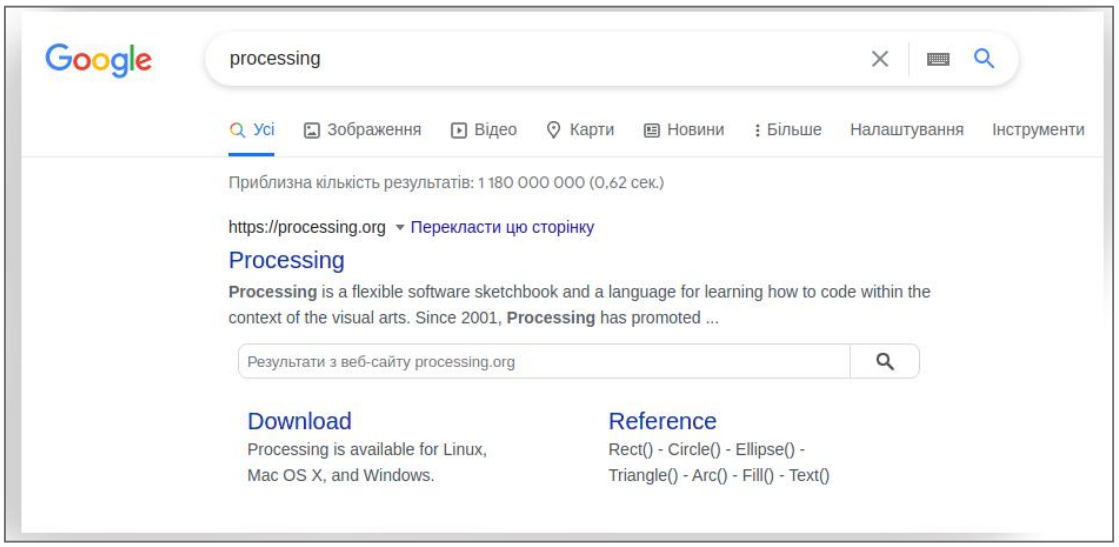
## Ознайомтесь з роботами інших в OpenProcessing

Ви можете переглянути програми-ескізи, створені іншими, і навіть редагувати їх, щоб побачити, як змінюється їх робота у вебсередовищі OpenProcessing. Це чудове місце для розміщення власних розробок, якими можна поділитися з іншими, у ньому ви можете переглянути програми, створені за допомогою Processing, воно дає вам можливість протестувати якийсь простий код і побачити результат його роботи навіть без необхідності завантажувати та встановлювати середовище програмування на власному комп’ютері.

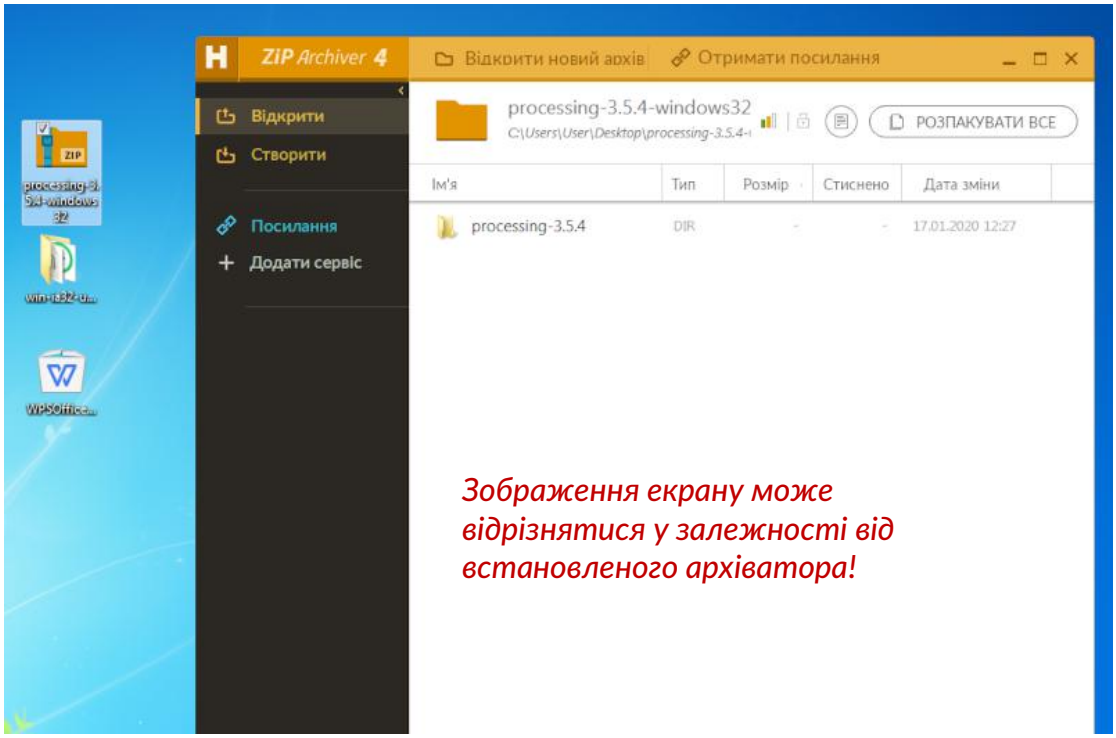
# Просто зануртесь: встановіть Processing

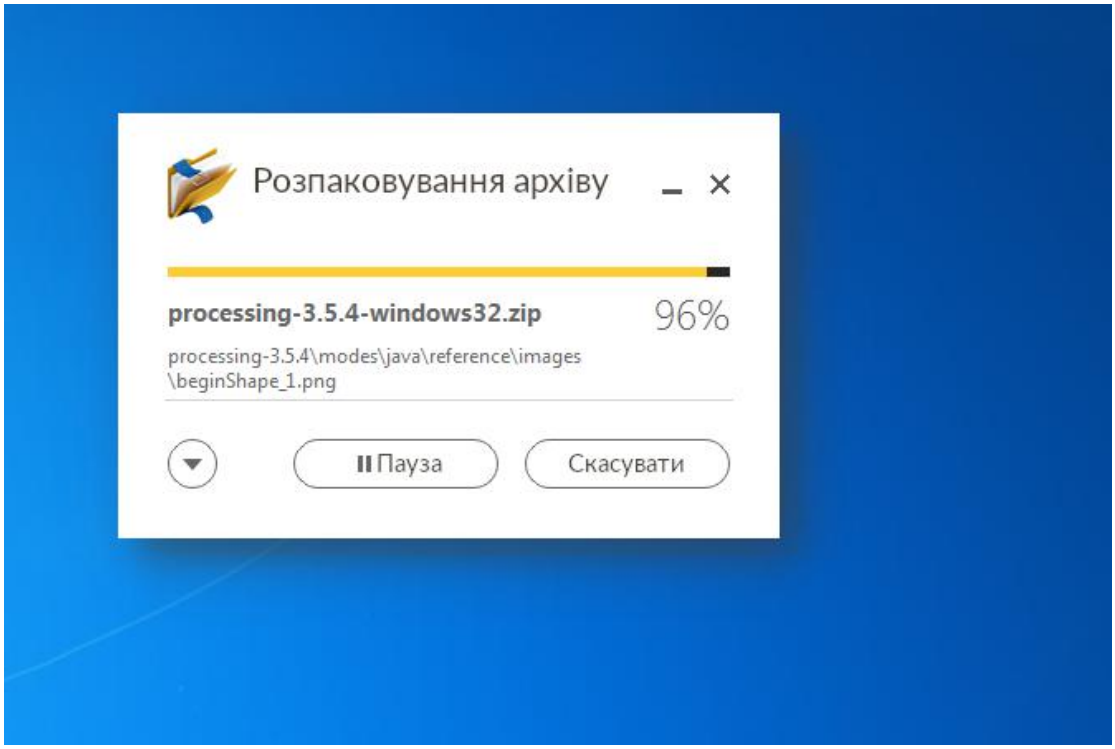
Найкращий спосіб досягти успіху в чомусь - це негайно почати робити активні кроки (якими б крихітними вони не були). Для початку роботи з Processing вам не потрібно нічого особливого, окрім комп'ютера з підключенням до Інтернету під управлінням Windows, Linux або Mac. Вам не потрібні ніякі базові знання з програмування.

Готові програмувати власні ескізи для Processing? Перейдіть на сторінку завантаження Processing.org і виберіть версію для своєї операційної системи (є версії для Windows, Mac OS та Linux).

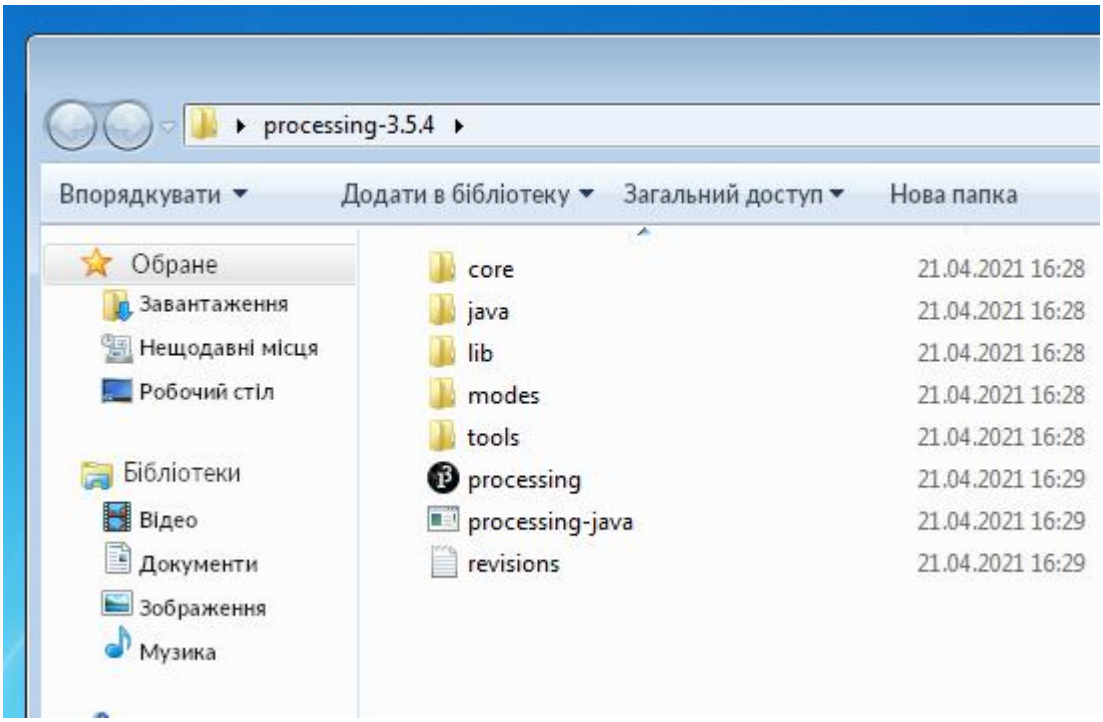



Видобудьте файли з архіву в окрему папку та відкрийте її.



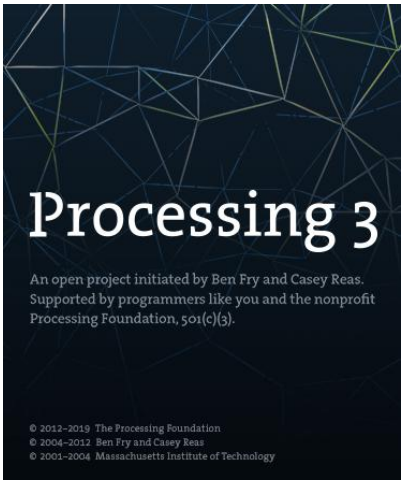



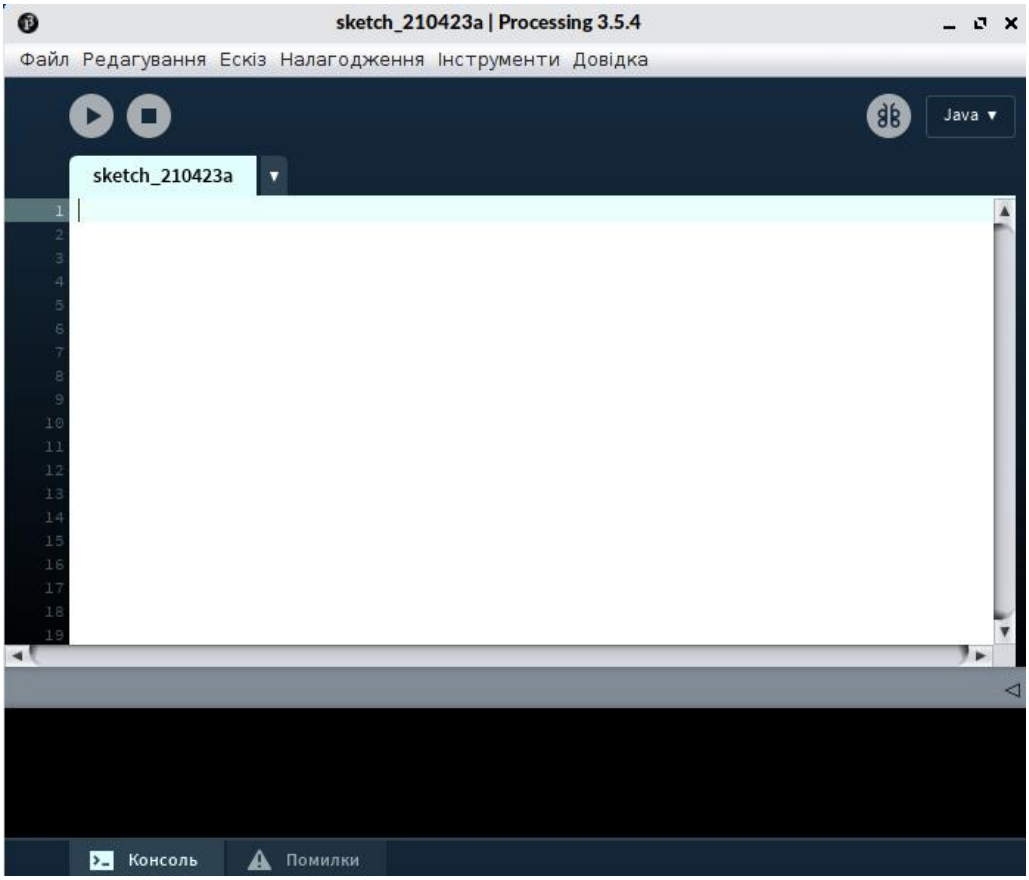
Якщо у вас 64-розрядна версія Windows, після відкриття папки з вмістом архіву ви побачите набір файлів, який виглядає приблизно так:




Для запуску середовища програмування скористайтесь файлом з логотипом Processing:

Якщо все пройшло успішно, після вікна з заставкою програми, ви потрапите в інтегроване середовище програмування (IDE), яке дозволить вам створювати власні програм-ескізи та запускати їх на виконання.





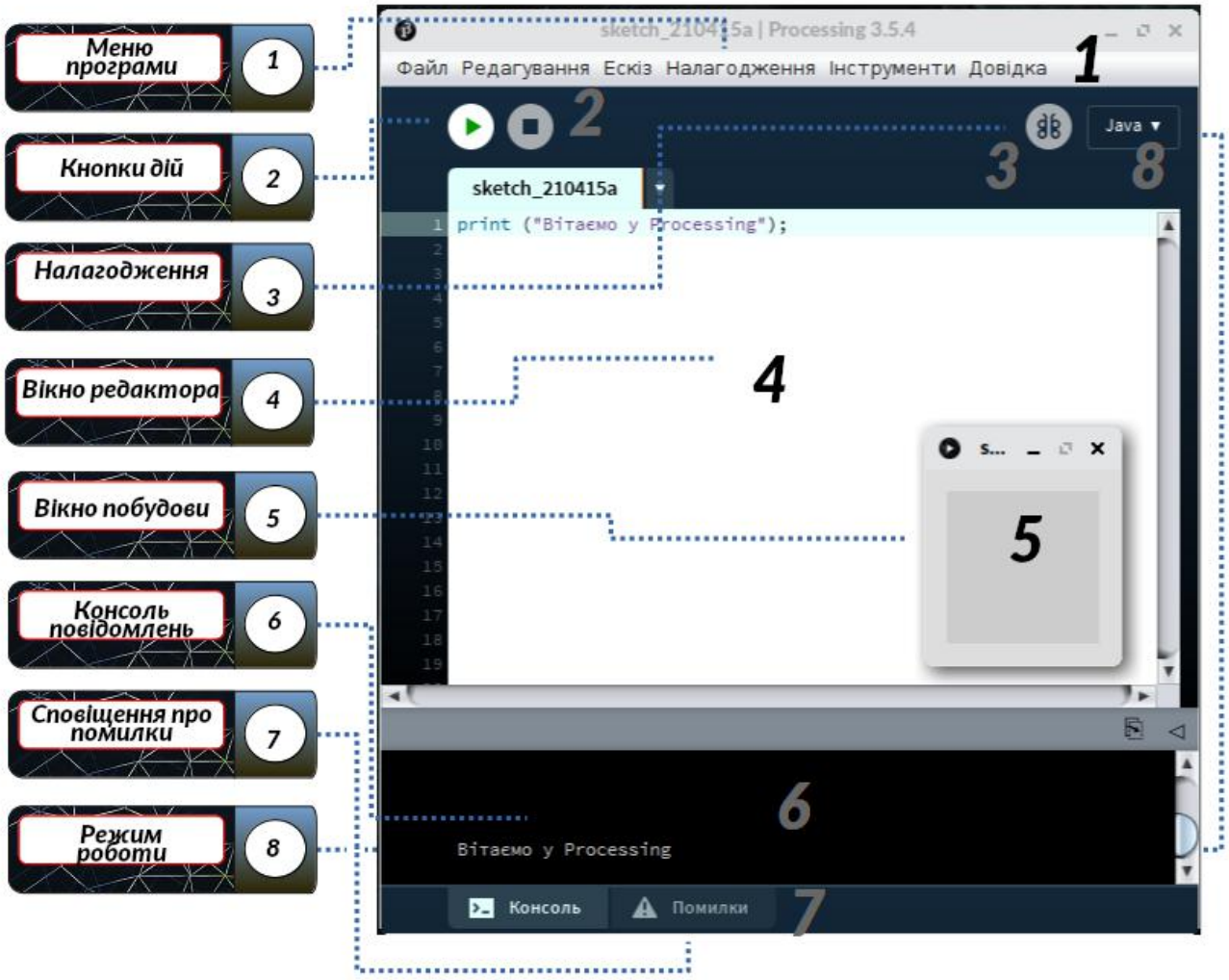

## Інтерфейс

На сьогоднішній день Processing перекладено українською мовою, хоча деякі повідомлення про помилки можуть виводитися на екран англійською мовою.

Інтерфейс користувача Processing складається з двох різних вікон: головного вікна, в якому ви будете створювати свій проєкт, і вікна перегляду побудови, в якому відображаються результат роботи програми (малюнки, анімація, відео).

Інтерфейс містить такі елементи:

1. Меню програми
2. Кнопки дій
3. Кнопка для активації режиму налагодження
4. Область редагування коду програми
5. Вікно побудови (область візуалізації зображень)
6. Консоль, яка включає вкладку для повідомлень під час роботи програми
7. Вкладка сповіщень про помилки.
8. Список режимів роботи.




## Кнопки дій

Кнопка “Запустити": запускає виконання вашого скетчу(вашої програми).

Кнопка "Зупинити": зупиняє виконання вашого скетчу.

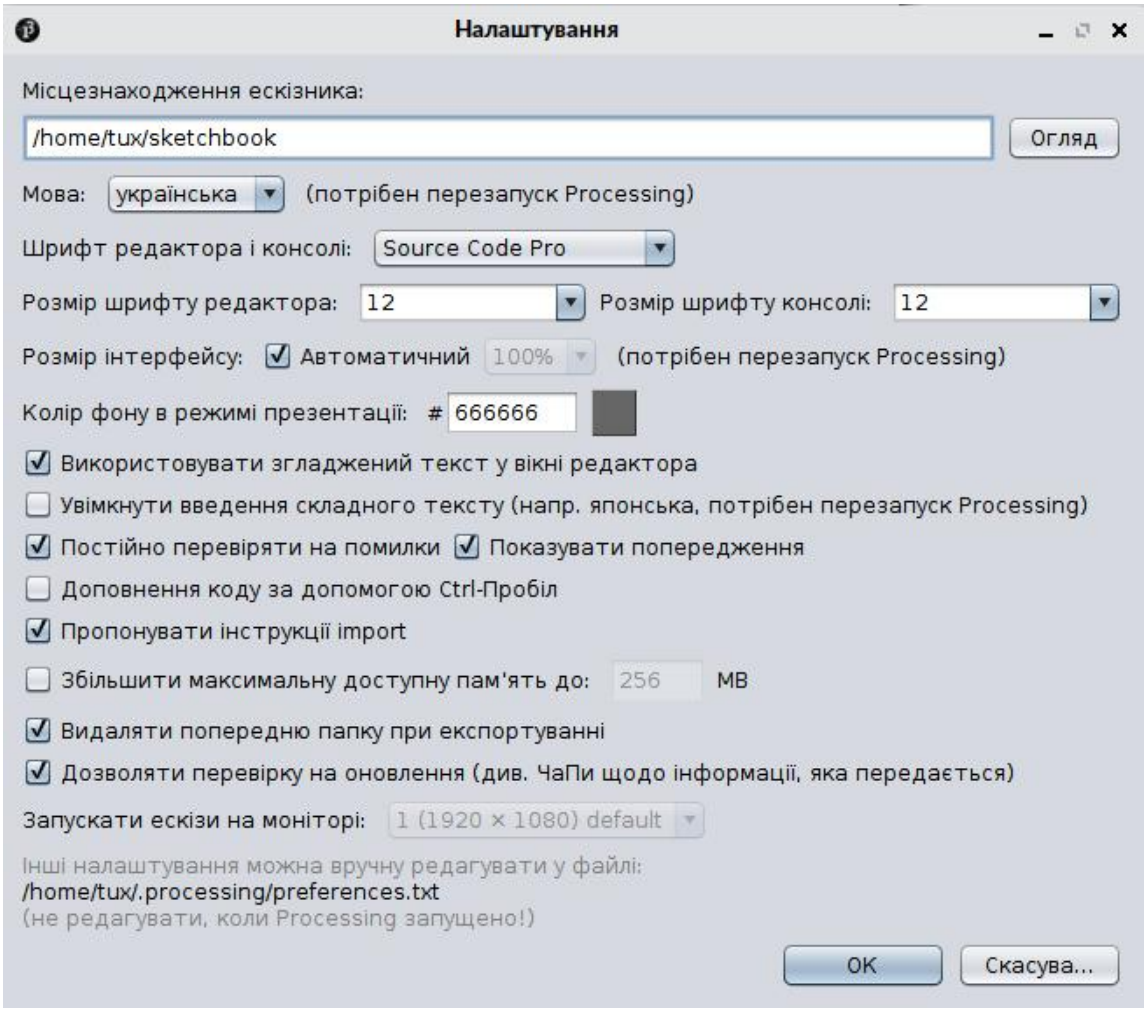
Processing дозволяє працювати в різних режимах, створюючи програми, специфічні для кожної цільової платформи (застосунок Java, для Android, P5js, Python, тощо.). Цими режимами можна керувати - ви можете будь-коли змінити цей режим з інтерфейсу, попередньо зберігши свій скетч.

## Робоча тека

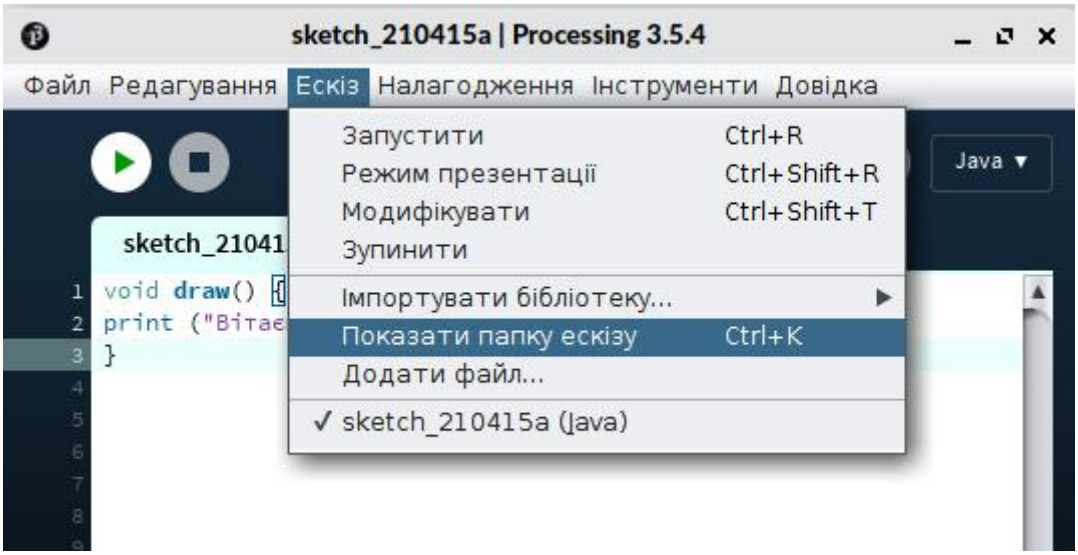
Це тека, в якій зберігаються скетчі та бібліотеки(зовнішні модулі, що пропонують додаткові функціональні можливості). За замовчуванням ця тека називається Processing(Processing) і знаходиться в Documents(на Mac)

або My Documents(на Windows). У GNU/Linux він знаходиться у вашому домашньому каталозі під назвою sketchbook .

Щоб змінити цю теку, перейдіть до меню Processing> Налаштування. У діалоговому вікні, що з'явиться, натисніть кнопку Огляд, щоб вибрати потрібну теку.




За необхідності у будь-який час ви можете дізнатись, якою є ваша робоча тека, вибравши у меню Ескіз > Показати теку ескізу. Цей параметр також доступний за допомогою комбінації клавіш ctrl+K у Windows/Linux або cmd+K на Mac.


sketch_210415a | Processing 3.5.4
Файл Редагування Ескіз Налагодження Інструменти Довідка
Java ▼
sketch_21041
1 void draw() {
2 print ("Вітає
3 }
Запустити Ctrl+R
Режим презентації Ctrl+Shift+R
Модифікувати Ctrl+Shift+T
Зупинити
Імпортувати бібліотеку...
Показати папку ескізу Ctrl+K
Додати файл...
✓ sketch_210415a (Java)

# Основи Processing

Processing пропонує як середовище розробки (Integrated Development Environment - IDE), так і набір додаткових бібліотек, які значно розширюють можливості програмного забезпечення. Середовище розробки дозволяє створювати (писати) програми, які в Processing називаються скетчами або ескізами, перетворювати їх у автономні, тобто незалежні від середовища розробки, виконувані файли, публікувати їх, виявляти та виправляти помилки. Воно містить основні функції для програмування, одночасно просте у використанні.

Як вже було зазначено, Processing базується на можливостях мови програмування Java, синтаксис, тобто сукупність правил якої, буде використовуватисяпід час програмування. Processing полегшит вам оволодіння цією мовою, дозволяючи виконувати відносно складні операції, такі як управління вікнами, звуком, відео, 3D та багатьма іншими, у простий та зрозумілий спосіб. Цей розділ знайомить вас з основами інтерфейсу Processing та мінімальними поняттями синтаксису Java, які ви повинні знати для початку.

## Основи мови

Програма на Processing складається з окремих вказівок (ми їх називатимемо інструкціями, командами, методами), які виконуються послідовно, якщо інший порядок не вказано у програмі. Processing створено на базі мови програмування Java і ця мова використовується для виконання вашої програми, яку ваш комп'ютер виконує після натиску на кнопку «Пуск». Мова програмування має певну кількість правил синтаксису, які, якщо їх не дотримуватись, заважатимуть правильному виконанню програми. Існує також ряд основних понять, які потрібно знати.

### Великі та малі літери

Processing чутливий до регістру, тобто він розрізняє великі та малі букви в іменах змінних та вказівок: РЗЕ відрізняється від рЗе!

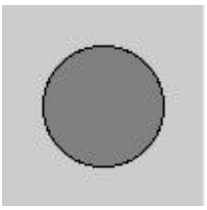
### Крапка з комою

В кінці кожної інструкції (малювати коло, провести обчислення тощо), ви повинні поставити символ “;” для того, щоб позначити кінець команди. У наведеному нижче прикладі ми використовуємо символи "//", щоб вставити коментар (пояснення), який буде проігноровано під час виконання(наявність коментарів у вашій програмі полегшує розуміння її роботи).

```
// Малюємо коло
ellipse(10,10, 10, 10);

// Створення змінної
int number = 10 + 23;
```

## Виклики функцій

Processing пропонує велику кількість заздалегідь визначених функцій, які називаються **методами**: малювати прямокутник, визначити колір, обчислити квадратний корінь тощо. Кожен із цих методів має певну назву, щоб використати їх, вам потрібно у текст програми записати їх ім'я, розрізняючи великі та малі літери, та вставити дужки після імені - іноді вам доводиться вказувати певні значення всередині дужок(колір, положення, розмір тощо). У наведеному нижче прикладі показано побудову сірого круга.

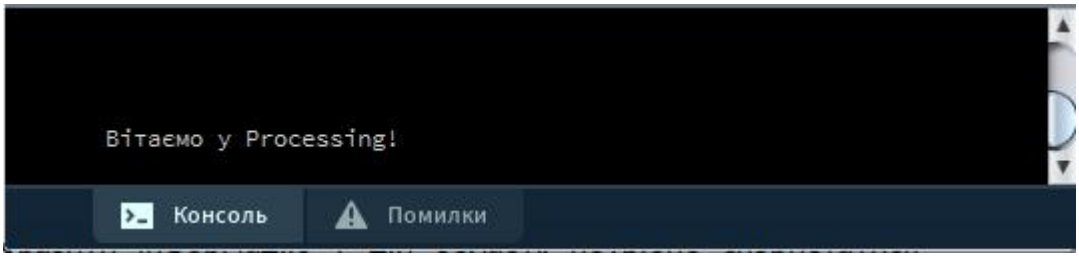


```
fill(128);
ellipse(50, 50, 60, 60);
```

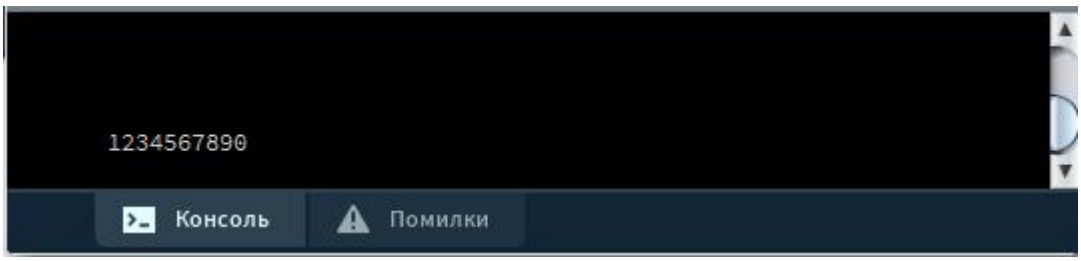
## Інформація на консолі

Консоль - текстове вікно, яке використовується для відображення повідомлень, у тому числі і для тестування та налагодження програми, які доречні для пошуку і виправлення помилок у ній. Щоб відобразити інформацію у цій області, потрібно скористатися методом `println()`;

```
println("Вітаємо у Processing!");
```

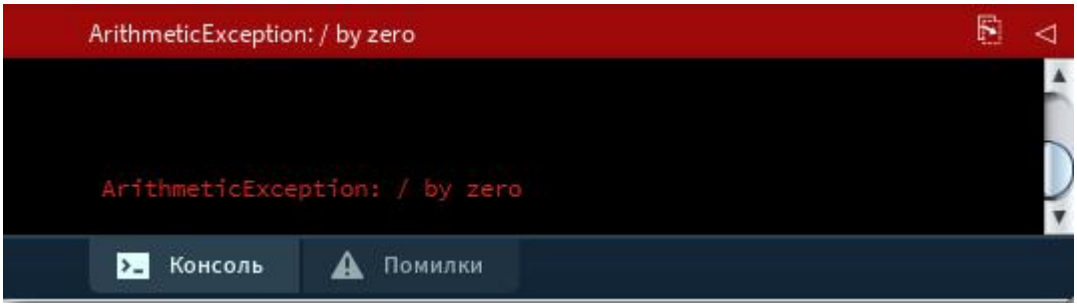



```
println(1234567890);
```




Спеціальна вкладка консолі використовується для сповіщень про помилки, які виникають під час роботи програми.

```
println(5/0);
```




## Арифметичні дії

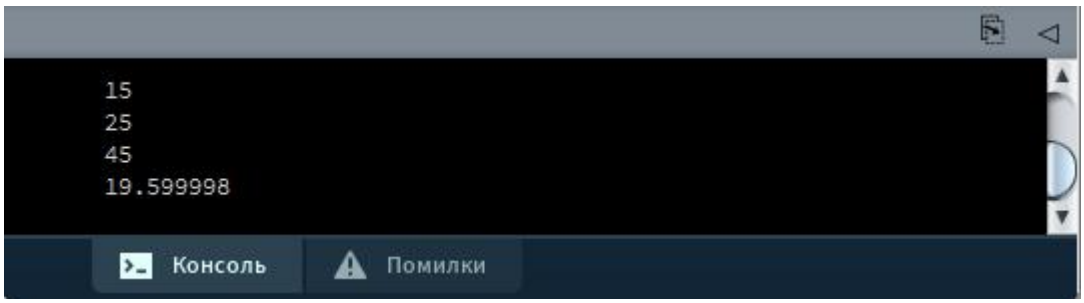
Processing дозволяє проводити математичні обчислення, адже під час виконання програми може виникнути необхідність обчислювати певні значення. Під час обчислень можна використовувати дії додавання, віднімання, множення та ділення, які можна поєднувати відповідно до потреби, також можна використовувати дужки для визначення порядку виконання дій.

**Пам'ятайте, що в Processing у десяткових дробах дробова частина відділяється від цілої крапкою, а не комою!**

Ось кілька прикладів арифметичних дій:

```
println(10 + 5);
println(10 + 5 * 3); // 5 * 3(тобто 15), потім додаємо 10
println((10 + 5) * 3); // 10 + 5(тобто 15), потім помножте
15 на 3
println(10.4 + 9.2);
```

Ця серія інструкцій, введена у вікні редагування Processing, дасть у консолі такий результат:




Поєднання деяких арифметичні дій з операцією присвоювання можна записувати у скороченому вигляді. Наприклад, `i++` дає той самий результат, що і `i = i + 1`. Так само `x + = 10` може замінити вираз `x = x + 10`.

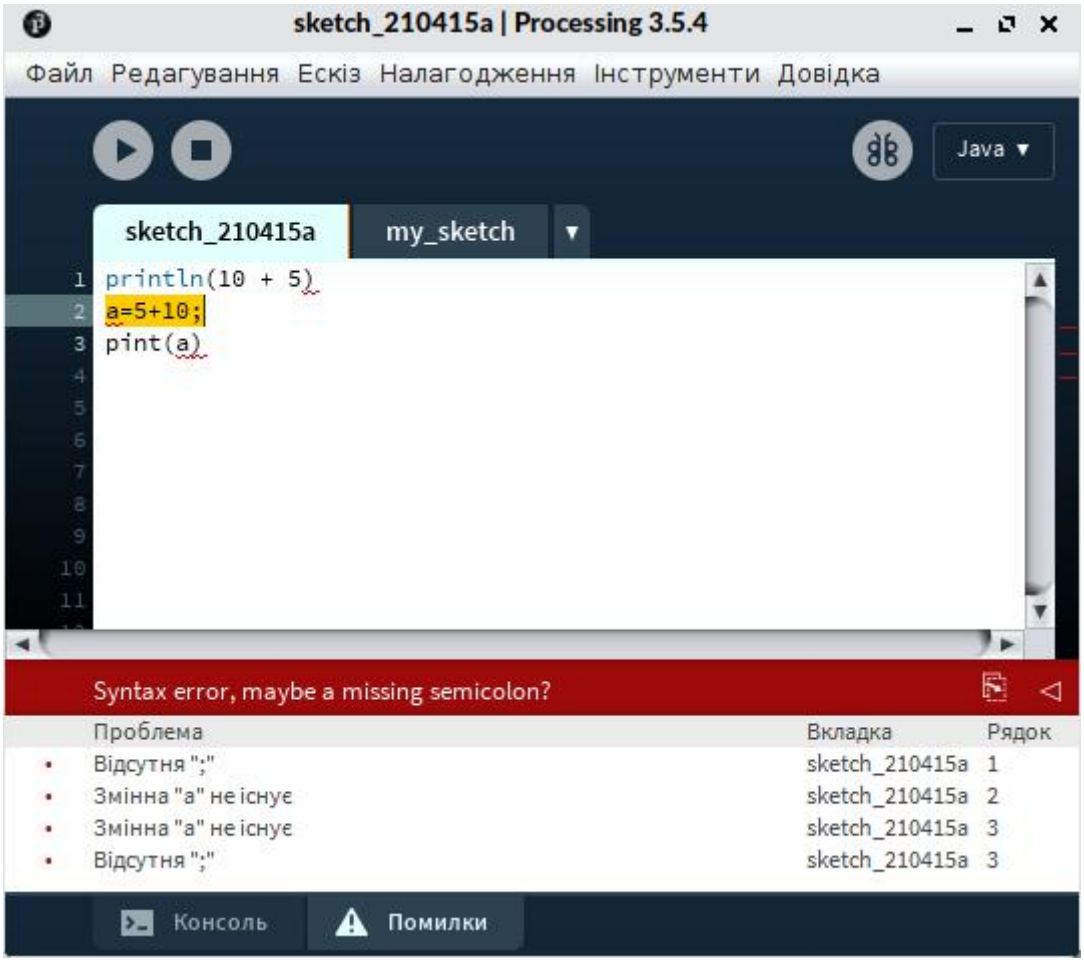
## Відображення помилок

Сучасні версії Processing мають перевірку синтаксису під час введення коду, причому неможливо запустити на виконання програму, яка має помилки.

Ці помилки вказуються двома різними способами:

- підкресливши червоним кольором проблемні елементи в редакторі.
- підсумовуючи всі помилки в консолі на вкладці «Помилки», вказавши характер проблеми та вкладку коду, в якій вона з'являється. Подвійне клацання помилки приведе вас у редакторі до місця, де Processing виявив цю помилку.

Іноді виправлення однієї помилки може каскадно вирішити інші помилки, пов'язаних з цією помилкою. Тому важливо виправити помилки в тому порядку, в якому вони були вказані, особливо якщо вони "близькі" в редакторі коду.

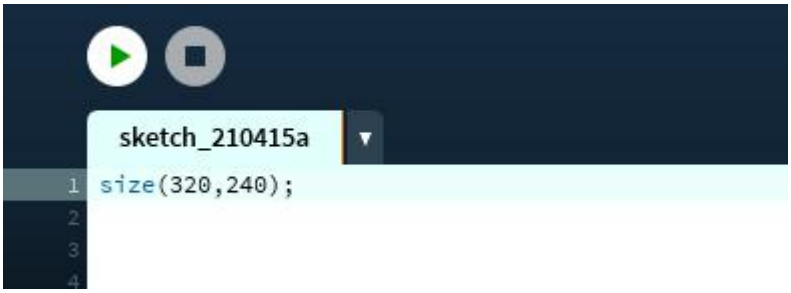


# Починаємо малювати

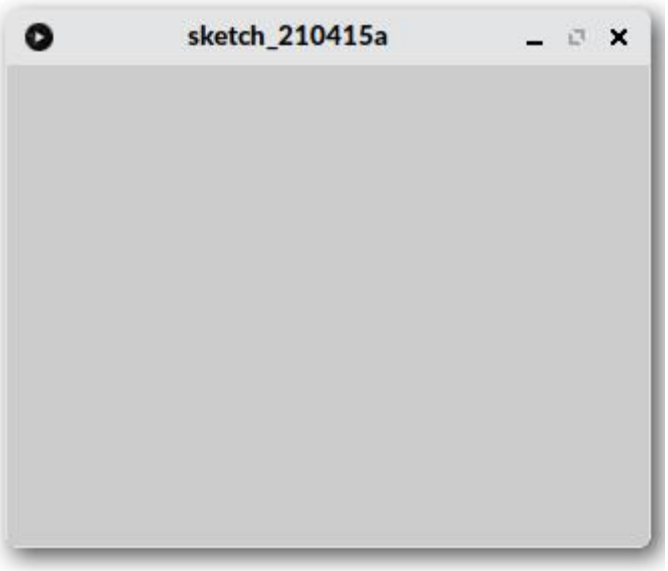
Простір для малювання є місцем, де візуалізується результат роботи програми. У цьому вікні виконуються побудови та виведення зображень.

Цей простір створюється під час виконання інструкції **`size()`**, яка має два аргументи: **`size(ширина, висота);`**.

Спробуйте, ввівши у вікні редагування IDE Processing таку команду:




Потім клацніть на кнопку «Запустити» - з'явиться створюється вікно візуалізації:

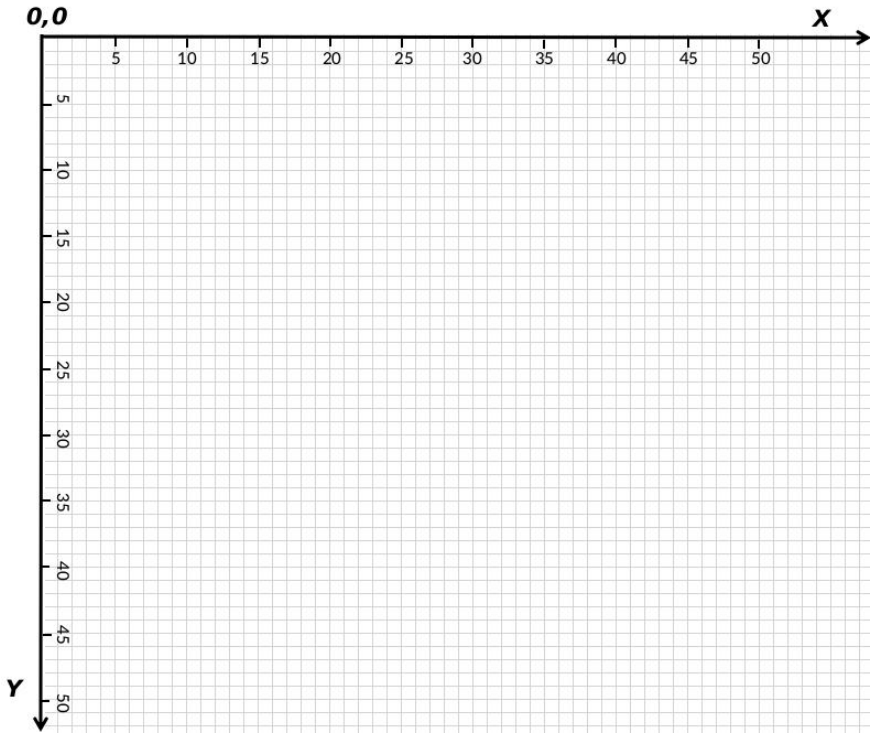



Дослідіть роботу цієї команди, змінюйте розміри цього вікна, змінюючи значення в дужках, щоб побачити результат.

Якщо розмір області побудови явно не вказано командою **`size`**, за замовчуванням буде використано вікно розміром 100 пікселів на 100 пікселів (піксель — найменший елемент зображення, кольорова точка зображення на екрані монітора).

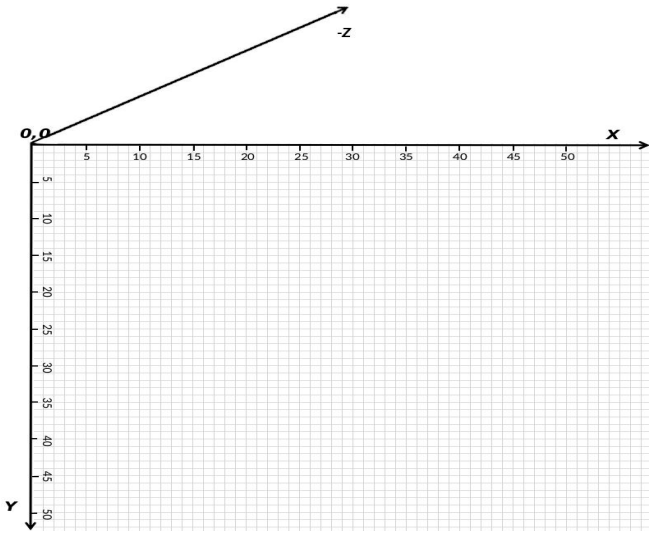
# Координати на площині та в просторі

Більшість графічних побудов ми будемо проводити на площині, тобто у двохвимірній системі координат(2D), у якій положення будь-якої точки можна записати двома числами, які відповідають проекції положення точки на осі координат OX (горизонтальна вісь) та OY(вертикальна вісь). На відміну від звичайної (декартової) системи координат, зазвичай ми не матимемо від'ємних значень координат, початок відліку знаходиться у верхньому лівому куті вікна візуалізації і він має координати x = 0 та y = 0. Значення x зростають зліва праворуч, а значення y збільшуються вниз. Максимальні значення x та y обмежуються розміром вікна візуалізації, хоча насправді вони можуть бути і більшими, але об'єкти з такими координатами не відображатимуться на екрані. у саме в цьому просторі ми будемо малювати.




Зауваження: положення початку відліку координат можна змінювати, скориставшись командою **`translate(x,y)`**, з якою ми познайомимось пізніше.

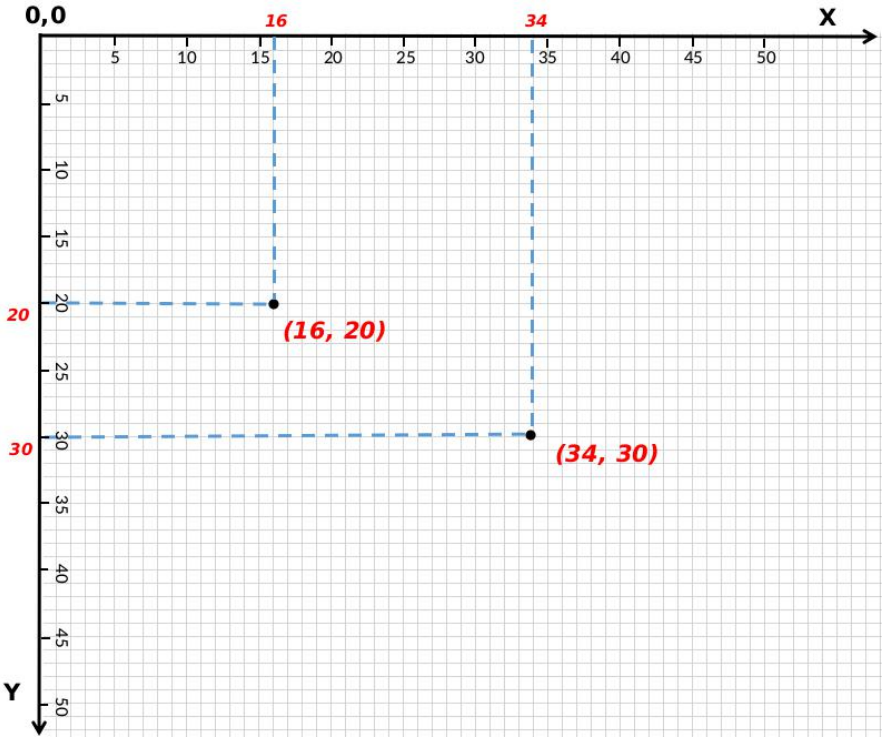
При роботі в 3-х вимірах(3D), крім двох координатних осей, існує третя координатна вісь Z, що виражає глибину:




Щоб вказати, що побудова буде виконуватись у тривимірному просторі(3D), використовується команда **`size`** з третім параметром: **`size(100, 100, P3D);`**

## Розмір області для малювання

Під час роботи програми ви можете будь-коли дізнатися розмір простору для малювання за допомогою зарезервованих слів **`width`** та **`height`**. Ці вказівки дуже корисні, коли потрібно, зокрема, побудувати фігури, розмір яких може адаптуватися до можливих змін у розмірі вікна побудови.

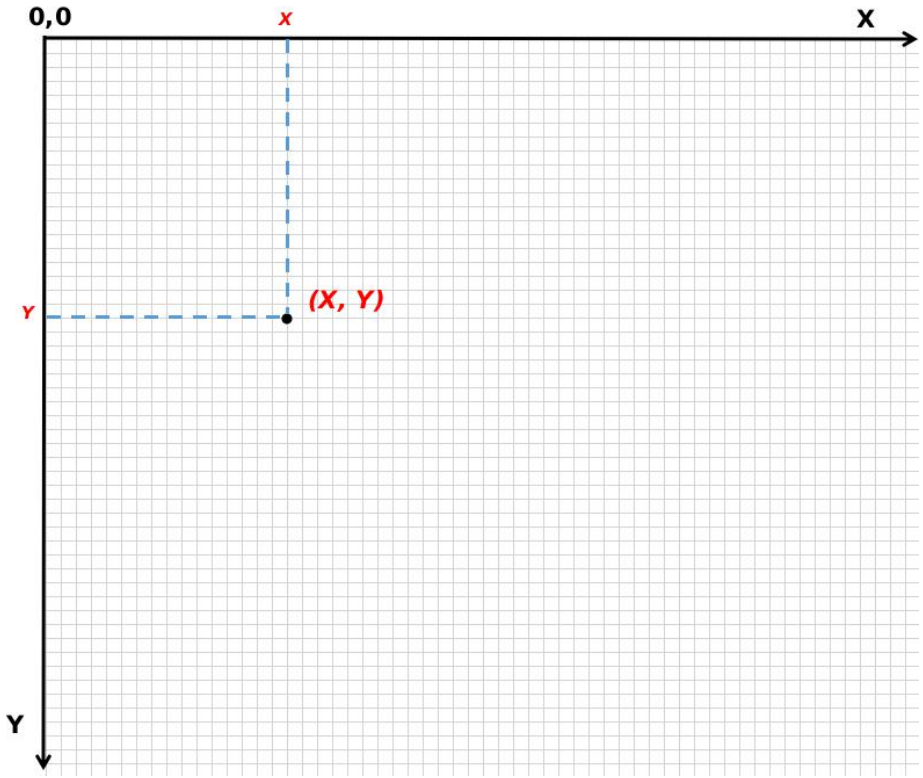


# Фігури

Processing надає зручний спосіб для побудових геометричних фігур, таких як точки, лінії, еліпси, багатокутники, криві та інші. Розглянемо основні з них.

## Точка

Починати малювати ми почнемо з побудови точки. На екрані точка - це еквівалент пікселя, розташованого у вікні побудови за двома координатами по осях **x** та **y**, що відповідають відповідно ширині(горизонтальній осі) та висоті(вертикальній осі) області побудови. Дотримуючись цього принципу, побудова точки в Processing здійснюється з використанням інструкції **point(x,y)**.



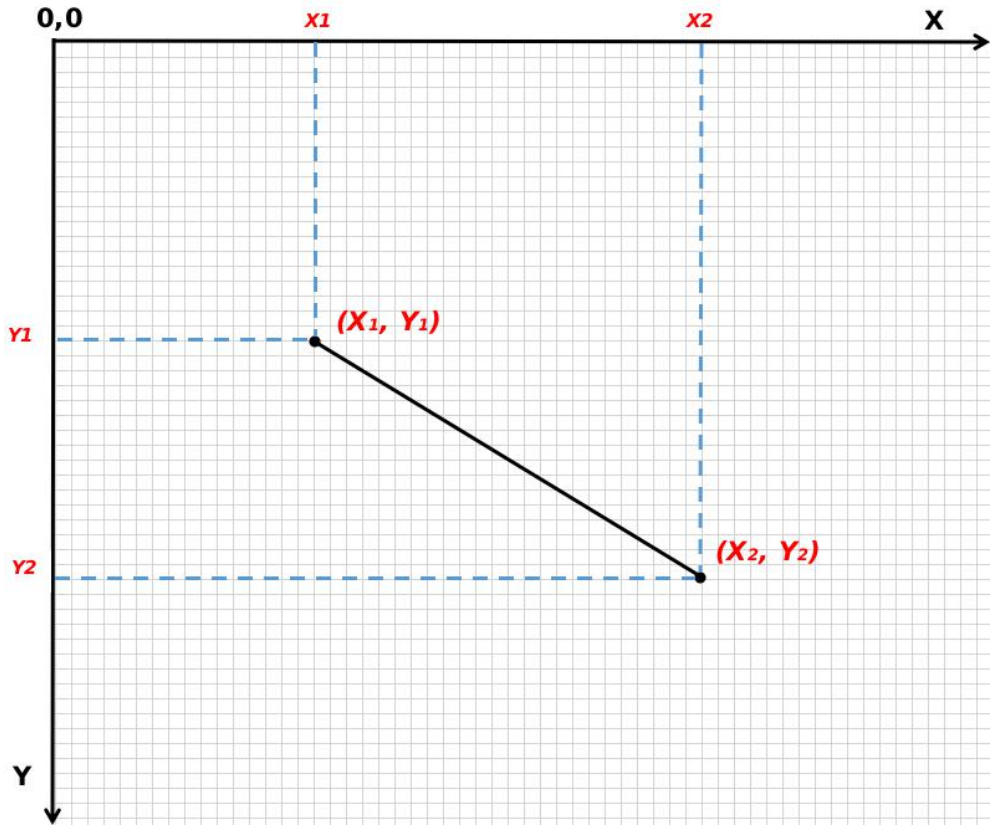

У наведеному прикладі точка дуже мала - вона розміщена в центрі вікна побудови.

```
point(50, 50);
```

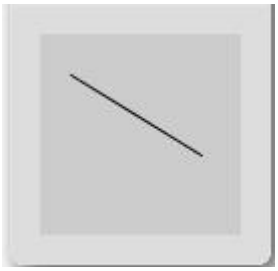
Зверніть увагу, що точка з координатами (50,50) розміщена в центрі вікна(простору для малювання), яке за замовчуванням має розмір 100x100. Змінюючи координати точки ми можемо розмістити її у будь-якому місці, якщо координати будуть більші розмір вікна побудови, то точка не відображатиметься.

## Лінія

За визначенням, пряма AB складається з множини точок між початковою точкою і кінцевою точкою B. Для її побудови нам потрібні лише координати **`x`** та **`y`** точок A і B. Отже, якщо, наприклад, у вікні за замовчуванням точка A знаходиться в області внизу ліворуч вашого вікна, а точка B - вгорі праворуч, наступні інструкції можуть побудувати цей рядок як **`line(x1,y1,x2,y2)`** :

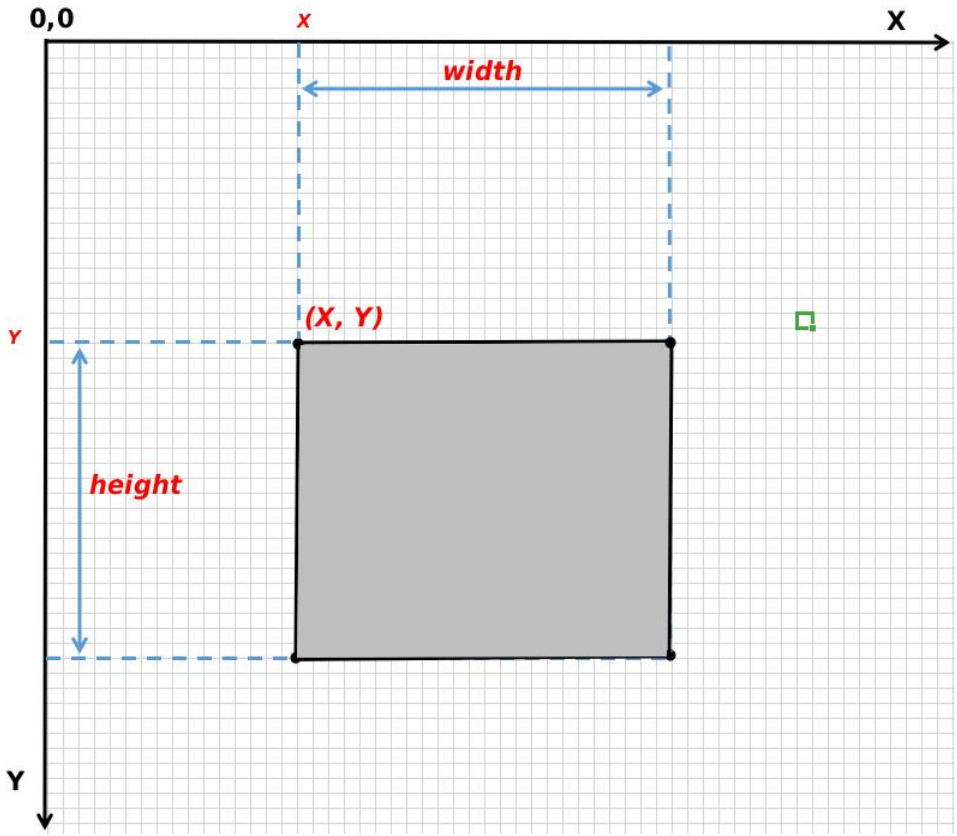



```
line(15, 20, 80, 60);
```



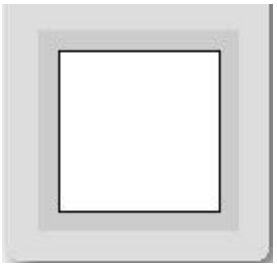
## Прямокутник

Прямокутник будується за чотирма значеннями - **`rect(x,y,ширина,висота)`**. Перша пара значень x і y, у режимі за замовчуванням(режим **`CORNER`**) відповідає лівому верхньому куту прямокутника, як і точка. З іншого боку, друга пара значень буде посилатися не на положення нижнього правого кута, а на ширину(на осі x, тобто по горизонталі) та на висоту(на осі y, по вертикалі) цього прямокутника.




Приклад:

```
rect(10, 10, 80, 80);
```



Оскільки останні два значення(ширина і висота) однакові, ми отримуємо квадрат. Дослідіть роботу програми, змінюючи значення та проаналізуйте результати.

Для того, щоб перша пара значень відповідала центруфігури (перетину двох діагоналей у координатах 50, 50) прямокутника, слід використовувати режим **CENTER** таким чином:

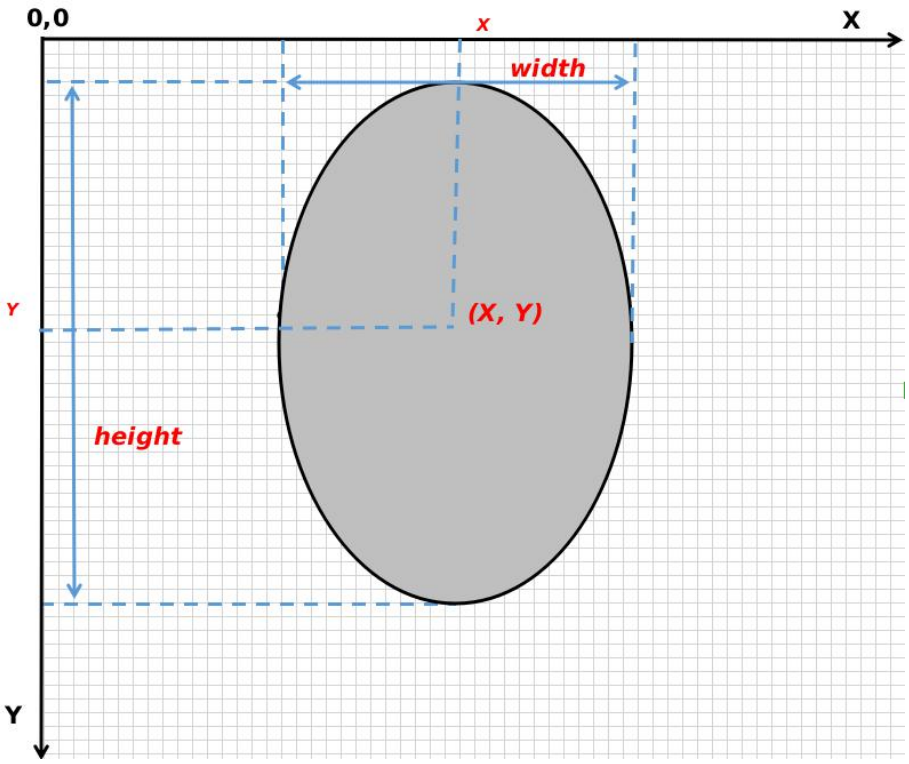
```
rectMode(CENTER);
rect(50, 50, 80, 80);
```

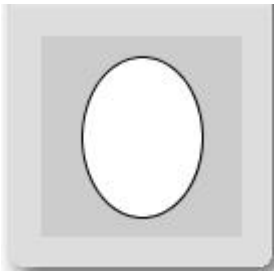
Це дає результат, ідентичний попередньому прикладу.

## Еліпс

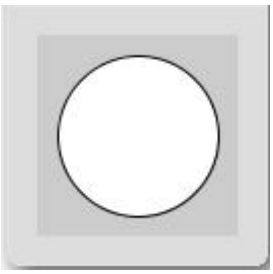
Як і прямокутник, еліпс(овал) будується в режимах **CENTER**(за замовчуванням), і **CORNER**. Наступна інструкція створює коло, центром якого є перші два значення в дужках. Третє значення відповідає розміру діаметра на горизонтальній осі(x), а четверте - розміру діаметра на вертикальній осі: зверніть увагу, що якщо 3-е і 4-те значення однакові, отримуємо коло, в іншому випадку — еліпс:

```
ellipse(50, 50, 60, 80);
```





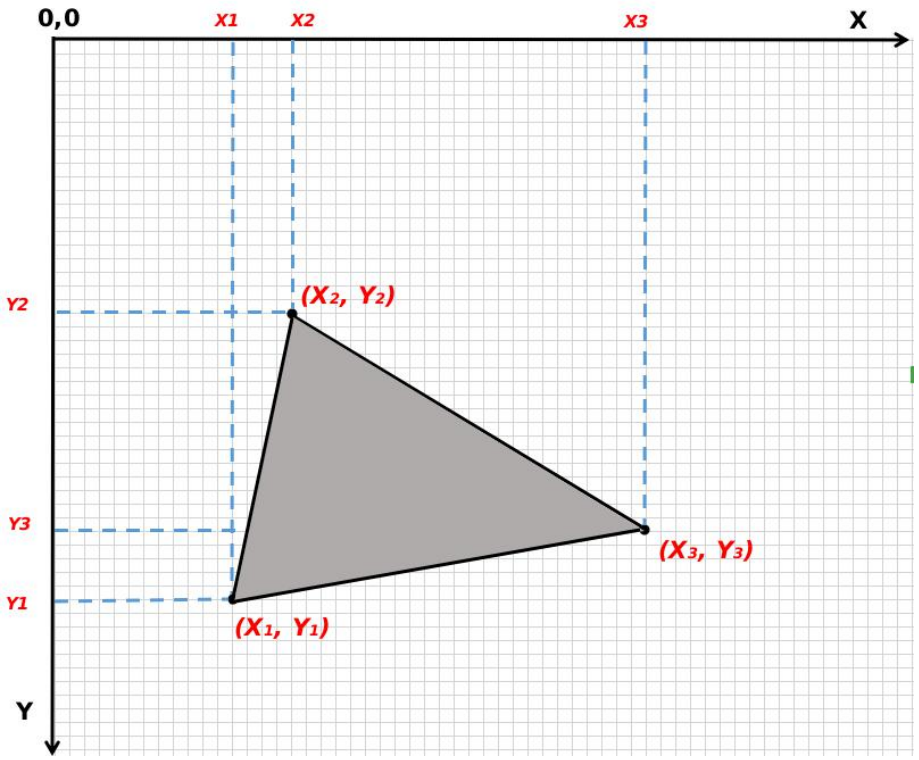
```
ellipse(50, 50, 80, 80);
```



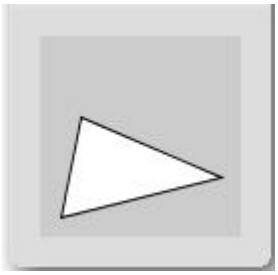
Дослідіть роботу програми, змінюючи 3-е і 4-те значення і спостерігайте за результатами.

## Трикутник

Трикутник - це фігура, яка має три точки-вершини. Виклик **`triangle(x1,y1,x2,y2,x3,y3)`** визначає три вершини цього трикутника:



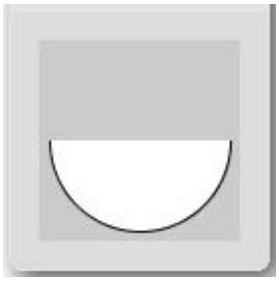
```
triangle(10, 90, 20, 40, 90, 70);
```



## Дуга

Дугу або сегмент круга можна побудувати, використавши вказівку **arc(x, y, ширина, висота, початок, кінець)**, де пара x, y визначає центр кола, друга пара - її розміри, а третя пара - початок і кінець кута дуги в радіанах:

```
arc(50, 50, 90, 90, 0, PI);
```

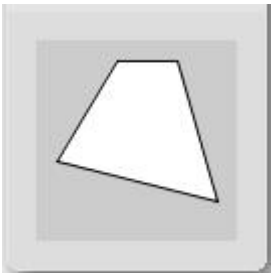


## Чотирикутник

Чотирикутник будується шляхом вказування координат x і y чотирьох вершин за годинниковою стрілкою:

**quad(x1,y1,x2,y2,x3,y3,x4,y4).**

```
quad(40, 10, 70, 10, 90, 80, 10, 60);
```



## Крива

Крива будується за допомогою команди **`curve(x1, y1, x2, y2, x3, y3, x4, y4)`**, де x1 та y1 визначають першу контрольну точку, x4 та y4 - другу контрольну точку, x2 та y2 - початкову точку кривої та x3, y3 - кінцеву точку кривої:

```
curve(50, 300, 10, 60, 90, 30, 50, 10);
```

## Крива Безьє

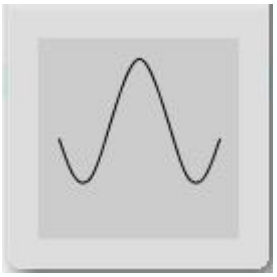
Криву типу Безьє можна побудувти з використанням інструкції **`bezier(x1,y1,x2,y2,x3,y3,x4,y4)`.**

```
bezier(10, 10, 80, 30, 30, 80, 90, 90);
```

## Плавна крива

Виклик **`curveVertex()`** малює кілька пар координат x та y між двома контрольними точками у формі, **`curveVertex(контрольна_точка_початку,xN,yN,xN,yN,xN,yN, контрольна точка кінця)`** яка дозволяє будувати плавні криві:

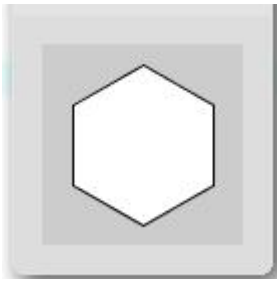
```
noFill();
beginShape();
  curveVertex(0, 0);
  curveVertex(10, 50);
  curveVertex(25, 70);
  curveVertex(50, 10);
  curveVertex(75, 70);
  curveVertex(90, 50);
  curveVertex(100, 0);
endShape();
```



## Полігони(багатокутники, ламані)

Кілька довільних фігур можна побудувати через координати їх вершин, використовуючи інструкції **beginShape(),vertex(x,y),...,endShape()**. Кожна точка будується за її координатами x та y. Функція CLOSE у **endShape(CLOSE)** вказує, що фігура буде закритою, тобто остання точка буде з'єднана відрізком з першою, як у наведеноми нижче прикладі побудови шестикутника:

```
beginShape();
   vertex(50, 10);
   vertex(85, 30);
   vertex(85, 70);
   vertex(50, 90);
   vertex(15, 70);
   vertex(15, 30);
endShape(CLOSE);
```



## Контури

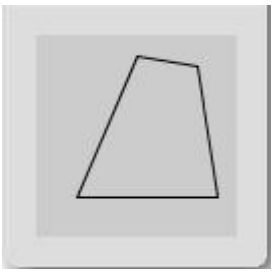
Ви помітили, що поки що фігури зі всіх пропонованих прикладів мають контур та заповнену кольором внутрішню область. Якщо ви хочете зробити контур невидимим, розмістіть вказівку **noStroke()** перед командою для побудови фігури:

```
noStroke();
quad(50, 10, 80, 15, 90, 80, 20, 80);
```

## Заповнення

При потребі можна побудувати фігури без заповнення внутрішньої частини, використавши інструкцію **`noFill()`** :

```
noFill();
quad(50, 10, 80, 15, 90, 80, 20, 80);
```



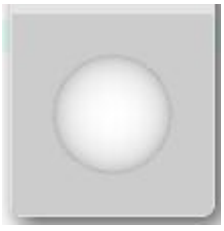
За замовчуванням фон області перегляду(простір для малювання) - нейтрально-сірий, контури фігур - чорні, а поверхня заливки - біла. У наступному розділі ви дізнаєтесь, як змінювати кольори на свій смак.

## 3D примітиви

Щоб побудувати фігури у тривимірному просторі(3D примітиви) спочатку необхідно скористатись командою **`size(x, y, P3D)`**, а вже потім використовувати інструкції **`sphere(радіус)`** для побудови сфери(кулі) і **`box(розмір)`** та **`box(довжина, висота, глибина)`** - у першому випадку будується куб із вказаною довжиною ребра, у другому - паралеллепіпед із вказаними розмірами. На тривимірних фігурах можна застосувати ефекти освітлення за допомогою **`lights()`**.

### Сфера

```
size(100, 100, P3D);
noStroke();
lights(); // підсвічуємо 3D-об'єкт
translate(50, 50, 0); // початок відліку перенесемо в центр
sphere(28);
```

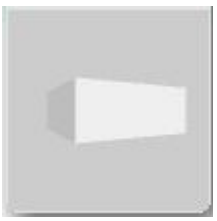


### Куб

```
size(100, 100, P3D);
noStroke();
lights();
translate(50, 50, 0);
rotateY(0.5);
box(40);
```

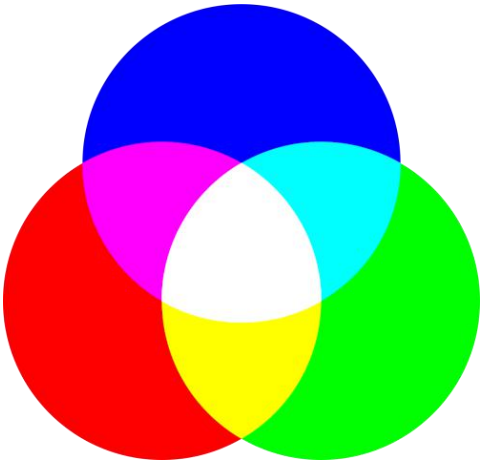
### Паралелепіпед

```
size(100, 100, P3D);
noStroke();
lights();
translate(50, 50, 0);
rotateY(0.5);
box(50, 20, 50);
```



# Кольори

Побудувати зображення на екрані означає певним чином змінити колір окремих пікселів- найменших точок екрану, кожна з яких може набути певного кольору. Кожен колір визначається трьома складовими - інтенсивністю червоного, зеленого та синього у різних пропорціях. Якщо «змішати» усі ці три кольори при максимальному значенні - отримаємо білий колір. Значення 0 кожного з цих трьох каналів дає чорний колір.



Зверніть увагу - чим більше значення кожного каналу, тим яскравішим є колір.

Для прикладу, 100% червоної складової, 80% зеленої та 20% синьої у результаті утворять помаранчевий колір. Метод **`fill()`** дозволяє вказати колір фігур, які будуть будуватися. Кожен канал кольору може набувати значень 0 до 255. Таким чином, 80% від 255 - це 204, а 20% від 255 - 51:

```
fill(255,204,51); // обрано помаранчевий колір
```

## Колір фону

Ви можете змінити колір фону, викликавши метод **`background()`**. **Попередження:** використання **`background()`** після того, як вже побудовано зображення, призведе до того, що побудоване зображення буде знищено!

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

## Колір заповнення

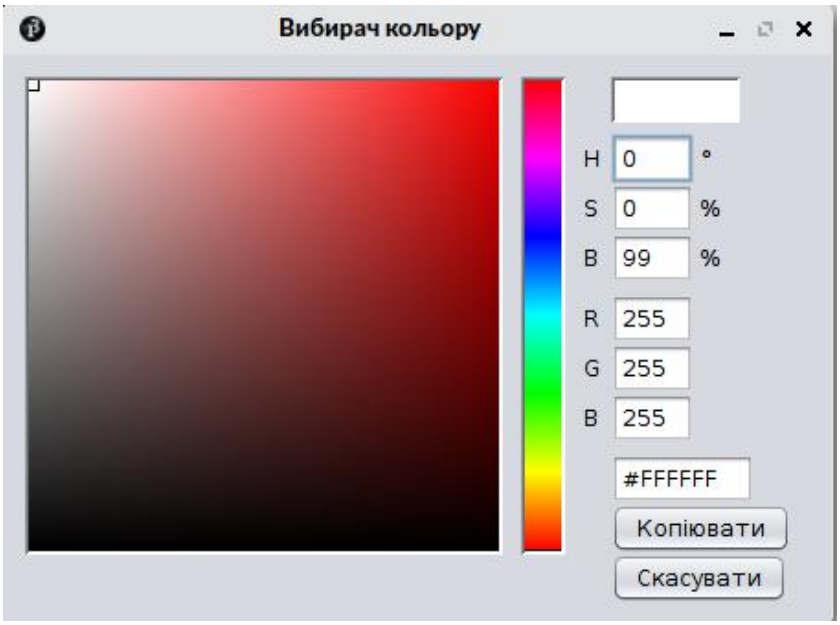
Щоразу, коли ми малюємо фігуру, ми робимо це з кольором заповнення, який вибрано на той момент. Змінити його можна викликавши метод **`fill()`**.

```
noStroke();
fill(255, 204, 51);
rect(25, 25, 50, 50);
```

Processing пропонує різні формати для вираження кольору. Якщо ви займаєтесь веб-програмуванням, ви, мабуть, знайомі з шістнадцятковим форматом. Відповідно до цього, один і той же оранжевий колір можна отримати також написавши:

```
fill(#ffcc33);
```

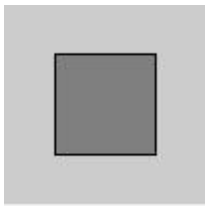
Числові значення кольорів запам'ятовувати не варто, адже середовище програмування надає інструмент "Вибирач кольору" (Інструменти > Вибрати колір):




Крім того, можна вказати значення альфа-каналу використовуваного кольору, тобто його ступінь прозорості. Для цього ми повинні вказати чотири параметри методу **`fill()`**. Четвертий аргумент - значення альфа, тобто прозорості.

```
noStroke();
fill(255, 204, 51); // помаранчевий
rect(25, 25, 50, 50);
fill(255, 255, 255, 127); // напівпрозорий білий
rect(35, 35, 50, 50);
```

Якщо ви хочете вибрати колір, який відповідає відтінку сірого, вам потрібно вказати лише один параметр методу **`fill()`**. Це значення сірого, від 0 до 255.



```
fill(127);
rect(25, 25, 50, 50);
```

Ви можете деактивувати заповнення фігур, викликавши метод **`noFill()`**.

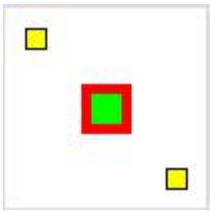
## Колір контуру

Щоб змінити колір контуру фігур, які ми малюємо, ми повинні викликати метод **`stroke()`** із параметрами каналів потрібного кольору. Виклик **`noStroke()`** вимикає колір контуру. Зображення маски клоуна створено з використанням цих двох інструкцій:

```
size(200, 200);
smooth();
background(255); // білий фон

stroke(#000000); // контур чорний
fill(#FFCC66); // заповнення жовте
strokeWeight(3);

translate(width/2, height/2);

ellipse(0, 0, 180, 100); // еліпс обличчя

stroke(255, 0, 0); // контур червоний
arc(0, 0, 90, 90, 0, PI); // дуга рота

fill(255,0,0);
ellipse(70, 0, 25, 25); // еліпс правого ока
ellipse(-70, 0, 25, 25); // еліпс лівого ока

fill(255, 255, 255); // заповнення біле
ellipse(28, -30, 25, 10); // еліпс правого ока
ellipse(-27, -30, 25, 10); // еліпс лівого ока

stroke(#000000); // контур наступних фігур буде чорним
fill(0, 0, 0); // контур чорний
ellipse(28, -30, 5, 5); // зіниця правого ока
ellipse(-27, -30, 5, 5); // зіниця лівого ока

noFill(); // наступні фігури не матимуть заповнення

bezier(-30, -50, -5, -60, -5, 0, -5, 20); // крива правої брови
bezier(30, -50, 5, -60, 5, 0, 5, 20); // крива лівої брови
line(-5, 20, 5, 20); // лінія носа для з'єднання кінця кривих
```

## Область дії зміни стилю зображення

За замовчуванням будь-яка зміна стилю(колір заповнення або контуру, ширина лінії або її вигляд) застосовуватиметься до всіх наступних побудов. Для окремих об'єктів можна деактивувати глобальні стилі, тимчасово застосувавши власні, попередньо використавши команду **`push()`** та команду **`pop()`** для повернення до глобальних стилів (**`pushStyle()`** та **`popStyle()`**.



```
size(100, 100);
background(255);

stroke(#000000);
fill(#FFFF00);
strokeWeight(1);
rect(10, 10, 10, 10);

push(); // Відкриваємо зміни стилю

stroke(#FF0000);
fill(#00FF00);
strokeWeight(5);
rect(40, 40, 20, 20);

pop(); // Закриваємо зміну стилю

rect(80, 80, 10, 10);
```

Вправа:

Видаліть команди **`push()`** і та **`pop()`** і спостерігайте за різницею у поведінці.

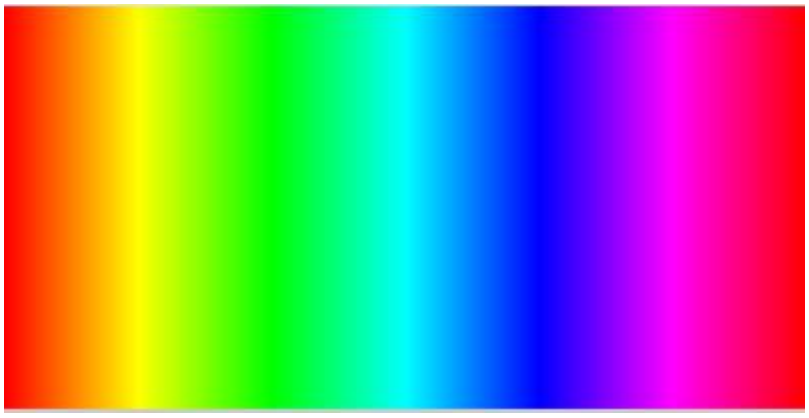
## Колірний простір

За замовчуванням Processing використовує кольорову модель RGB, яка визначає довільний колір як суміш трьох кольорових компонентів - червоного (Red), зеленого (Green) та синього (Blue), кожен з яких може набувати значення від 0 до 255, але за допомогою методу **`colorMode(RGB, 100)`** можна обрати шкалу значень компонентів від 0 до 99, що може бути доречним для ефективності перерахунку значень кольорів. Наприклад, виклик **`colorMode(RGB, 1.0)`** змінить масштаб, який ми використовуємо для вказівки кожного кольорового каналу, від нуля до одиниці.

Метод **`colorMode`** можна і для зміни використовуваної кольорової моделі, Processing також підтримує режим HSB - "відтінок, насиченість,

яскравість". Відтінок відповідає числу, що вказує положення кольору на хроматичній шкалі (червоний - ліворуч, потім помаранчевий, жовтий, зелений, синій та фіолетовий, як у веселці).

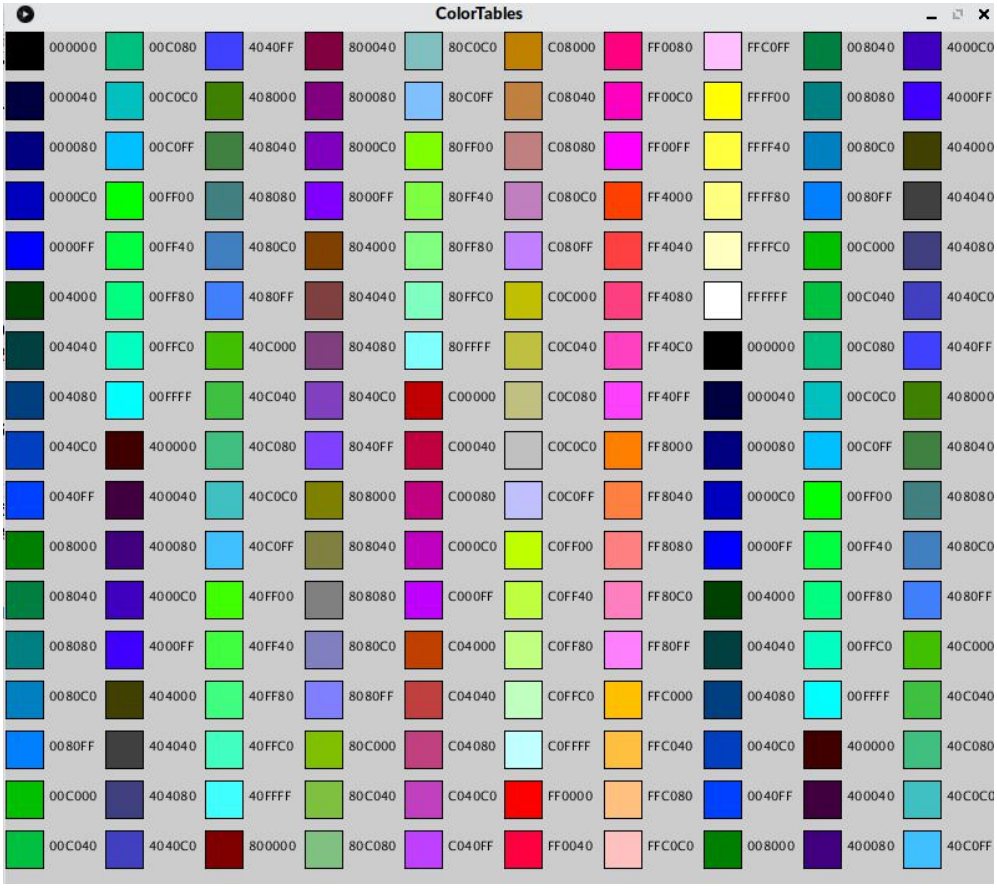
В наведених нижче ілюстративних прикладах використано команду повторення **`for()`**, яка буде розглядатися у наступних розділах. Тут ми використовуємо кольоровий режим HSB, щоб створити градієнт, схожий на кольори веселки.



```
size(400,200);
noStroke();
colorMode(HSB, 400,100,100);
for (int i = 0; i < 400; i++) {
    stroke(i, 128, 128);
   line(i,0,i,200);
}
```

Порівняйте з роботою цієї з програми з таким ж параметрами в режимі RGB:

```
size(400,200);
noStroke();
colorMode(RGB, 400,100,100);
for (int i = 0; i < 400; i++) {
    stroke(i, 128, 128);
   line(i,0,i,200);
}
```




**Таблиця кольорів RGB:**

```
size(800,800);
int R=0;
int G=0;
int B=0;
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 17; j++) {
    color c = color(R,G,B);
    push();
    textSize(10);
    fill(0,0,0);
    text(hex(c,6),i*80+35, j*40+16);
    pop();
    fill(R,G,B);
    if (B<256) {B=B+64;}
    else {B=0;if (G<256) {G=G+64;}
              else {G=0;if (R<256) {R=R+64;}
                        else {R=0;}
                   }
         }
    rect(i*80, j*40,30,30);
    }
}
```

# Текст

У вікні побудови ми можемо розміщувати будь-які символи або рядки символів таким же способом, як і використовувані раніше геометричні фігури, такі як лінії або еліпси, фактично ми «малюємо» текст за допомогою відповідної інструкції.

## Привіт, Processing!

Відкрийте нове вікно редагування, введіть наступний рядок і запустіть його, натиснувши кнопку запуску:

```
text("Привіт!", 10, 50);
```

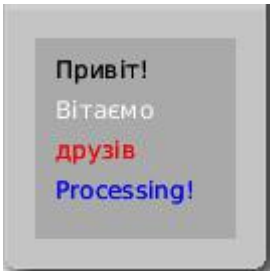
Ви отрмаєте вікно візуалізації розміром 100x100 пікселів із написом «Привіт!» білого кольору:


Інструкція **`text()`** створює графічне зображення текст у вікні побудови. Для цього потрібні три параметри: {повідомлення, яке ми хочемо написати}, {його координата x}, {його координата y}.

Колір тексту, як і будь-якої геометричної фігури, визначається методом **`fill()`**, який змінює колір заповнення:

```
fill(0);
text("Привіт!", 10, 20);

fill(255);
text("Вітаємо", 10, 40);

fill(255,0,0);
text("друзів", 10, 60);

fill(0,0,255);
text("Processing!", 10, 80);
```

В результат кожне слово буде «написано» окремим кольором:

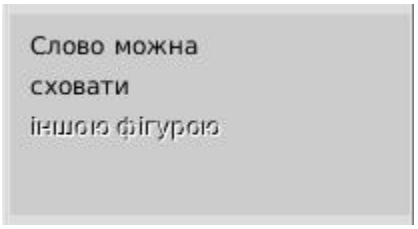



## Перекривання

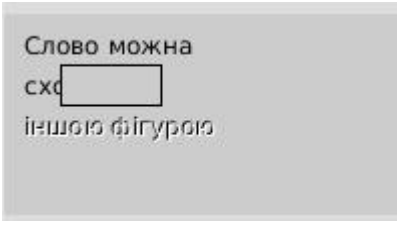
Як ми зазначили у вступі до цього розділу, слова вписані всередину вікна перегляду є елементом графічного зображення, а не текстом у звичному розумінні цього слова - ви не зможете безпосередньо виділити та копіювати цей текст за допомогою миші або клавіатури. Єдине, що можна зробити - знищити частину тексту, побудувавши фігуру на його місці за допомогою відповідних інструкцій. У поданому прикладі при натиску мишки частина тексту перекривається прямокутником:

```
void setup(){
size(200,100);
fill(0);
text("Слово можна", 10, 20);
text("сховати", 10, 40);

fill(0);
text("іншою фігурою", 10, 60);

fill(255, 255, 255);
text("іншою фігурою", 11, 61);
}
void draw() {         // необхідно для опрацьовування натиску миші
}
void mouseClicked() { // Дія при натиску кнопки миші
fill(204);
rect(28,25,50,20);    // будуємо прямокутник
}
```

До натиску мишки




Після натиску мишки



Варто пам’ятати, що при побудові і визначенні остаточного візуалізації є важливим порядок операцій - інструкції виконуються зверху вниз, і фігури, які побудовані пізніше, можуть перекривати фігури, побудовані раніше.