

Grafika Komputerowa – Materiały Laboratoryjne

Laboratorium 5 – Processing

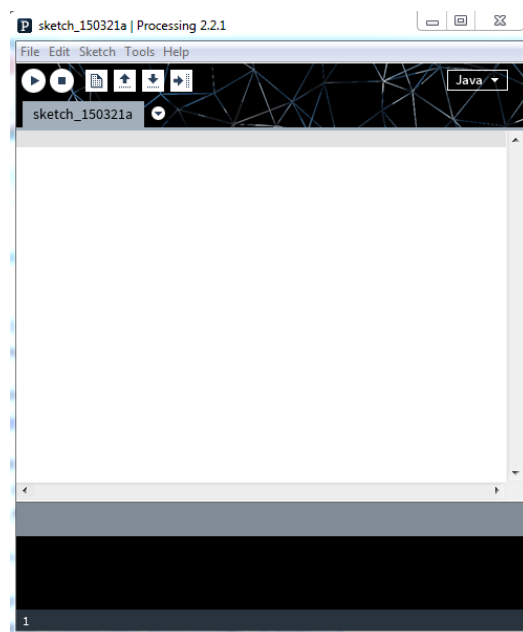
Wstęp


Laboratorium 5 obejmuje zagadnienie generowania i przetwarzania obrazów z wykorzystaniem języka Processing 3, dedykowanego do obróbki grafiki.

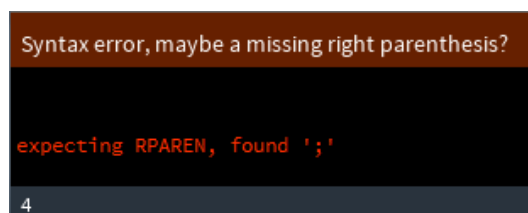
Parser i narzędzie developerskie dla języka Processing można pobrać z witryny [processing.org](https://processing.org/download/?processing), adres <https://processing.org/download/?processing>

Przed rozpoczęciem ćwiczenia należy uruchomić program processing.

Efektem będzie okno programu z obszarem pozwalającym na wprowadzanie kodu.



Programy w języku Processing określane są mianem Sketch a sam język przypomina uproszczoną wersję C. Sketche uruchamia się wciskając przycisk run . Dolny pasek służy do wyświetlania komunikatów o wykrytych błędach składni.



Przed rozpoczęciem ćwiczenia należy pobrać paczkę **lab_5_materialy.zip** z witryny miki.cs.pollub.pl i rozpakować ją do folderu Lab5 utworzonego na pulpicie

Ćwiczenie 1 – Zapoznanie z Processing

Zadanie: Narysuj podstawowe kształty

Narysuj prostokąt

Aby zlecić narysowanie prostokąta należy użyć komendy `rect(poz_x, poz_y, szerokość, wysokość)`. Np. `rect(100,100,300,150)`; Znakiem zakańczającym komendę jest standardowo średnik. Wykonanie powyższego kodu spowoduje wyświetlenie okienka z.. kwadratem

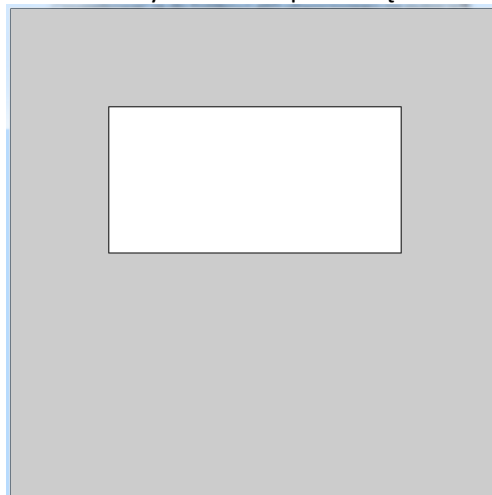


Processing wykorzystuje współrzędne x, y odpowiadające pikselom obrazu i liczone od lewego górnego narożnika okna.

Wykorzystana przez nas komenda powinna w teorii narysować prostokąt umieszczony w odległości x 100 i y 100 punktów od tego narożnika oraz szerokości 300 i wysokości 150. Otrzymaliśmy jednak kwadrat. Dzieje się tak z powodu braku zdefiniowania wielkości kanwy – okna w którym ma być narysowany prostokąt. Kod:

```
size(500,500);  
rect (100,100,300,150);
```

Powoduje już prawidłowe wyświetlenie prostokąta



Pozostaje zdefiniowanie kolorów linii, wypełnienia i tła.

Tło definiuje się poleceniem `background(r, g, b)`; gdzie r, g, b są wartościami 0-255 w systemie kolorów RGB (można również podać pojedynczą wartość poziomu szarości)

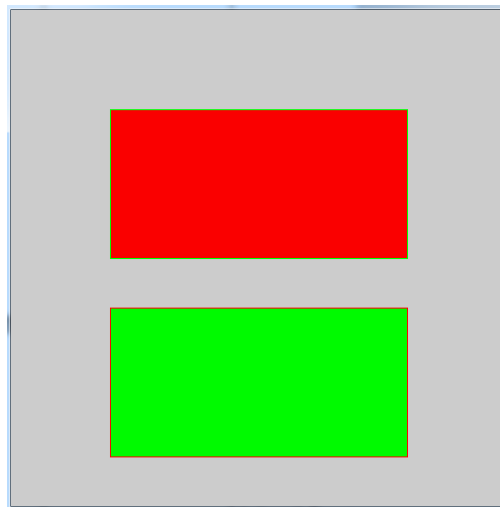
Analogicznie kolor linii obiektu to `stroke(r, g, b)`; a wypełnienie to `fill(r, g, b)`;,. Wartości te mogą być zmieniane ale zmiany obowiązują tylko dla nowo rysowanych obiektów. Pozwala to na tworzenie obiektów o różnych

kolorach.

Np. kod:

```
size(500,500);  
fill(250,0,0);  
stroke(0,255,0);  
rect(100,100,300,150);  
fill(0,250,0);  
stroke(255,0,0);  
rect(100,300,300,150);
```

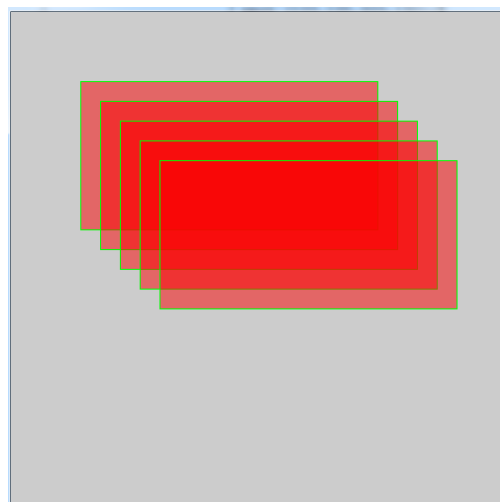
Da poniższy efekt



Określenie koloru dla powyższych poleceń może posiadać jeszcze 4-ty parametr definiujący stopień przezroczystości (r, g, b, alfa).

Poniższy kod ilustruje zastosowanie przezroczystości

```
size(500,500);  
fill(250,0,0,127);  
stroke(0,255,0);  
for (int i=1; i<6; i++){  
  rect(50+20*i,50+20*i,300,150);  
}
```





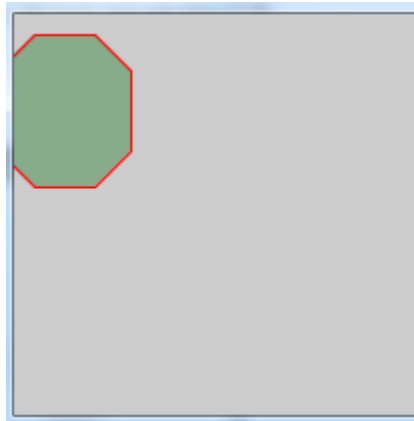
Zadanie samodzielne	
	<p>Narysuj widoczek „domek pod lasem”.</p> <p>Wykorzystaj również takie podstawowe kształty jak</p> <p>Punkt – point(x, y);</p> <p>Linia – line(x1, y1, x2, y2);</p> <p>Trójkąt - triangle(x1, y1, x2, y2, x3, y3);</p> <p>Czworokąt - quad(x1, y1, x2, y2, x3, y3, x4, y4)</p> <p>Elipsa – ellipse(x_środką, y_środką, szerokość, wysokość);</p> <p>Oraz definicję grubości linii – strokeWeight(n);</p>

Ćwiczenie 2 – Rysowanie dowolnych kształtów

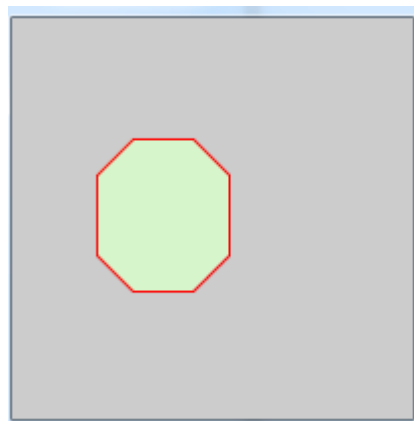
Zadanie: Narysuj wielokąt		
Narysuj wypełniony kolorem	ośmiokąt losowym	<p>Kroki:</p> <p>Losową wartość koloru można uzyskać stosując funkcję random()</p> <pre>fill(random(255), random(255), random(255));</pre> <p>Aby zlecić narysowanie dowolnego kształtu można zastosować rysowanie metodą „połącz punkty”. Program najpierw zapisuje kolejne punkty w pamięci a potem wykonuje ich połączenie. Ważna jest więc kolejność definiowania punktów. Należy również starać się NIE definiować punktów tak aby linie je łączące się przecinały. W takim przypadku Processing będzie miał kłopoty z wypełnieniem tego kształtu.</p> <p>Rozpoczęcie „nagrywania” punktów kształtu realizowane jest komendą beginShape();</p> <p>Każdy punkt (werteks) jest definiowany poleceniem vertex(x, y); „Nagrywanie” zakańcza się po dotarciu do komendy endShape(); Może ona zawierać parametr endShape(CLOSE); który spowoduje domknięcie kształtu (narysowanie linii między ostatnim i pierwszym werteksem)</p> <p>Kod naszego programu:</p> <pre>size(200, 200); fill(random(255), random(255), random(255)); stroke(255, 0, 0); beginShape(); vertex(10, 10); vertex(40, 10); vertex(58, 28); vertex(58, 68); vertex(40, 86); vertex(10, 86); vertex(-8, 68); vertex(-8, 28);</pre>



endShape(CLOSE);

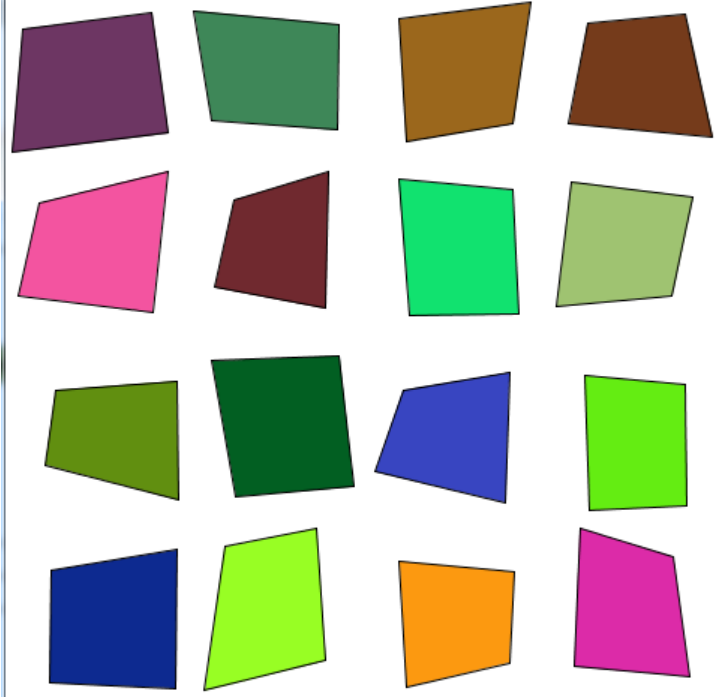


Jak widać w trakcie ręcznego wyliczania pozycji werteksów przekroczyliśmy krawędź okna kanwy. Aby naprawić sytuację możemy zamiast przeliczania współrzędnych punktów przesunąć układ współrzędnych: `translate(x, y)`; Efekt jest widoczny dla wszystkich nowo narysowanych elementów, dlatego musimy umieścić to polecenie przed rozpoczęciem rysowania kształtu.



Zadanie samodzielne

Wykorzystaj poznane funkcje rysowania kształtów do narysowania zestawu czworokątów o losowym wypełnieniu i narożnikach odchylających się losowo o maksimum $\pm 10\%$ od kształtu kwadratu

	 <p><i>W wykonaniu zadania przyda się pętla for</i></p>
Zadanie: Narysuj figurę z gładkimi zakrzywieniami	
Narysuj kształt przypominający płatek kwiatu	<p>Processing posiada kilka komend pozwalających na rysowanie zakrzywionych kształtów. Znamy już ellipse, nie pozwala ona jednak na tworzenie dowolnie zakrzywionych linii.</p> <p>Jedną z dostępnych opcji jest funkcja curve tworząca linię krzywą zgodnie ze schematem</p> <pre>curve(cpx1, cpy1, x1, y1, x2, y2, cpx2, cpy2);</pre> <p>gdzie oprócz początku i końca krzywej podane są dwa punkty kontrolne cpx1, cpy1 oraz cpx2, cpy2</p> <p>punkty kontrolne regulują kształt krzywej zgodnie z zasadą: Styczna do krzywej w jej punkcie początkowym x1, y1 jest równoległa do linii łączącej pierwszy punkt kontrolny cpx1, cpy1 z końcem krzywej x2, y2. Analogicznie styczna do krzywej w jej punkcie końcowym x2, y2 jest równoległa do linii łączącej drugi punkt kontrolny cpx2, cpy2 z początkiem krzywej x1, y1.</p>



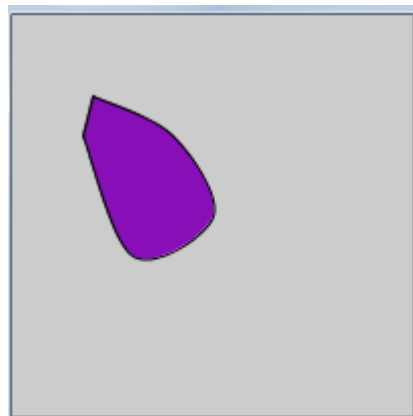
Źródło: processing.org

Pojedyncza krzywa to nadal za mało aby móc rysować dowolne gładkie kształty. Do tworzenia wielocłonowych krzywych wykorzystuje się polecenie `curveVertex(x,y)`; występujące w układzie `beginShape()` / `endShape()`;

Pierwszy i ostatni z serii punktów `curveVertex()` są traktowane jako punkty kontrolne. Pozostałe punkty kontrolne dla werteksów określających kolejne krzywe są wyliczane automatycznie według zasady utrzymania gładkości linii. Często jest definiowanie punktów kontrolnych początkowego i końcowego w tych samych miejscach co początek i koniec krzywej

```
size(200, 200);
fill(random(255), random(255), random(255));

stroke(0);
beginShape();
curveVertex(40, 40); // pierwszy punkt kontrolny
curveVertex(40, 40); // pierwszy punkt krzywej
curveVertex(80, 60);
curveVertex(100, 100);
curveVertex(60, 120);
curveVertex(35, 60); // ostatni punkt krzywej
curveVertex(35, 60); // ostatni punkt kontrolny
endShape(CLOSE);
```



**Zadanie samodzielne**

Wykorzystaj poznaną metodę tworzenia zamkniętych gładkich kształtów do stworzenia rysunku czterolistnej koniczyny

Ćwiczenie 3 – Transformacje**Zadanie: Narysuj koniczynkę**

Narysuj czterolistną koniczynkę wykorzystując kształt pojedynczego listka

Kroki:

Przeliczanie współrzędnych dla kolejnych listków jest dość uciążliwe. (W przypadku koniczyny było to jeszcze w miarę proste – zamiany x/y) Zdecydowanie lepszym rozwiązaniem jest wielokrotne wykorzystanie zestawu współrzędnych z pierwszego listka.

Znamy już jedno z poleceń transformacji układu współrzędnych – przesunięcie `translate(x,y)`;

Dwa kolejne to obrót względem początku układu współrzędnych – `rotate(rad)`; oraz skala – `scale(s)`; Kąt obrotu podawany jest w radianach

Dodatkowo dostępne są dwa polecenia uproszczające transformacje wielu obiektów.

`pushMatrix()`; zapisuje na stosie bieżący stan układu współrzędnych, `popMatrix()`; przywraca układ współrzędnych do pierwszego stanu ze stosu. Pozwala to na wykonanie transformacji układu współrzędnych w celu wykonania określonego przekształcenia rysowanego obiektu a następnie powrót układu współrzędnych do stanu pierwotnego.

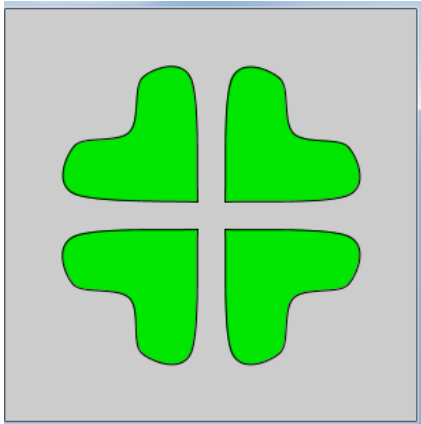
Algorytm rysowania 4 listnej koniczyny wyglądałby więc następująco

```
size(300, 300);
fill(0, 230, 0);
smooth();
stroke(0);
translate(150,150);

beginShape();
curveVertex(10, 10); // pierwszy liść
curveVertex(10, 10);
curveVertex(100, 15);
curveVertex(100, 50);
curveVertex(60, 60);
curveVertex(50, 100);
curveVertex(15, 100);
curveVertex(10, 10);
curveVertex(10, 10);
endShape(CLOSE);

pushMatrix();
rotate(radians(90));
beginShape();
curveVertex(10, 10); // drugi liść
curveVertex(10, 10); // obrócony o 90 stopni
curveVertex(100, 15);
curveVertex(100, 50);
curveVertex(60, 60);
```




	<pre>curveVertex(50, 100); curveVertex(15, 100); curveVertex(10, 10); curveVertex(10, 10); endShape(CLOSE); popMatrix(); pushMatrix(); rotate(radians(180)); beginShape(); curveVertex(10, 10); // trzeci liść curveVertex(10, 10); // obrócony o 180 stopni curveVertex(100, 15); curveVertex(100, 50); curveVertex(60, 60); curveVertex(50, 100); curveVertex(15, 100); curveVertex(10, 10); curveVertex(10, 10); endShape(CLOSE); popMatrix(); pushMatrix(); rotate(radians(270)); beginShape(); curveVertex(10, 10); // czwarty liść curveVertex(10, 10); // obrócony o 270 stopni curveVertex(100, 15); curveVertex(100, 50); curveVertex(60, 60); curveVertex(50, 100); curveVertex(15, 100); curveVertex(10, 10); curveVertex(10, 10); endShape(CLOSE); popMatrix();</pre> 
Zadanie samodzielne	
	Zbuduj układ czterech koniczynek o różnym kolorze ułożone jedna na drugiej według zasady: każda następna koniczynka jest obrócona o 30 stopni i zmniejszona o 25% w stosunku do poprzedniej

Ćwiczenie 3 – Funkcje i animacja

Zadanie: Narysuj kilka koniczynek



Narysuj koniczynki różnej wielkości i w różnych miejscach	<p>Do tego zadania przydatne byłoby stworzenie funkcji rysującej liść koniczyny.</p> <p>Processing wykonuje skrypty – Sketche w dwóch trybach. Dotąd korzystaliśmy z trybu statycznego (static), gdzie kod jest wykonywany raz zgodnie z kolejnością linii. W tym trybie Processing nie umożliwia wykorzystania funkcji</p> <p>Processing może również wykonywać skrypty w trybie aktywnym (active). W trybie tym istnieje podział na dwa obszary – funkcje wykonywania komend skryptu.</p> <p>Pierwszy obszar objęty strukturą <code>setup() { }</code> zawiera komendy wykonywane raz na początku aktywacji programu</p> <p>Drugi obszar objęty strukturą <code>draw() { }</code> zawiera komendy wykonywane cyklicznie co klatkę aż do zakończenia działania programu.</p> <p>Istnienie obszaru <code>setup()</code> aktywuje tryb aktywny, obszar <code>draw()</code> nie jest obowiązkowy, jeśli nie potrzebujemy odświeżania zawartości ekranu</p> <p>Definicję funkcji tworzy się w analogiczny sposób jak <code>setup</code> i <code>draw</code></p> <p><code>void Nowa_funkcja(parametr1, parametr2, ...) { kod funkcji }</code></p> <p>i umieszcza poza tymi dwoma obszarami</p> <p>Przykładowy kod Sketch w trybie active realizujący zadanie z ćwiczenia 2.</p> <pre>void setup() { size(600, 600); background(255); fill(150); stroke(0); for (int i = 0; i < 4; i = i+1) { for (int a = 0; a < 4; a = a+1) { plotRandomizedQuad (i*120+20, a*120+20, 80, 80, 0.2, 0.2); } //end for i } //end for a } // end setup</pre> <pre>void plotRandomizedQuad(float x, float y, float w, float h, float randW, float randH) { float jitterW = w*randW; float jitterH = h*randH; int xmid = round(x+w/2); int ymid = round(y+h/2); fill(random(255),random(255),random(255)); beginShape(); vertex(x+random(-jitterW, jitterW), y+random(-jitterH, jitterH)); vertex(x+random(-jitterW, jitterW), y+h+random(-jitterH, jitterH)); vertex(x+w+random(-jitterW, jitterW), y+h+random(-jitterH, jitterH)); vertex(x+w+random(-jitterW, jitterW), y+random(-jitterH, jitterH)); endShape(CLOSE); } // end plotRandomizedQuad</pre>
--	---



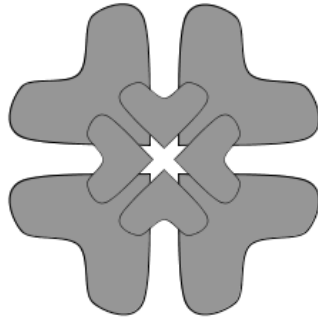
Funkcja rysowania koniczyny powinna mieć parametry odpowiadające za pozycję obrót i skalę koniczynki.

koniczynka(x, y, R, s);

Funkcja rysowania liścia powinna mieć parametry pozwalające na obrót rysuj_lisc(r);

```
void setup() {  
    size(600, 600);  
    background(255);  
    fill(150);  
    stroke(0);  
  
    koniczynka(200, 200, 0, 1);  
    koniczynka(200, 200, 45, 0.5);  
}  
  
void koniczynka(int x, int y, int R, float s){  
  
    pushMatrix();  
    translate(x,y);  
    rotate(radians(R));  
    scale(s);  
    lisc(0);  
    lisc(90);  
    lisc(180);  
    lisc(270);  
    popMatrix();  
}  
  
void lisc(int r){  
  
    pushMatrix();  
    rotate(radians(r));  
    beginShape();  
    curveVertex(10, 10); // pierwszy punkt kontrolny  
    curveVertex(10, 10); // pierwszy punkt krzywej  
    curveVertex(100, 15);  
    curveVertex(100, 50);  
    curveVertex(60, 60);  
    curveVertex(50, 100);  
    curveVertex(15, 100);  
    curveVertex(10, 10); // ostatni punkt krzywej  
    curveVertex(10, 10); // ostatni punkt kontrolny  
    endShape(CLOSE);  
    popMatrix();  
  
}
```



	
	Zmodyfikuj kod tak aby możliwe było definiowanie koloru i przezroczystości rysowanej koniczynki
Zadanie samodzielne	
	<p>Wykorzystaj informacje przedstawione w ćwiczeniu 3 do zbudowania animacji rysowania kolejnych płatków stokrotki (ze zmianą koloru) oraz jej późniejszego obracania</p> <p>Hint: Pewne podpowiedzi można znaleźć w tym tutorialu: https://processing.org/tutorials/transform2d/</p>