

# Dvodimenzionalna grafika

Mathematica ima bogato zbirko funkcij za delo z grafičnimi objekti. V tej lekciji bomo spoznali, kako v Mathematici delamo z dvodimenzionalno grafiko tako, da bomo napisali funkcije za tlakovanje evklidske in hiperbolne ravnine.

## Dvodimenzionalni grafični elementi

Osnovni dvodimenzionalni elementi v Mathematici so:

<b>Point</b> [{ $x$ , $y$ }]	točka s koordinatama ( $x$ , $y$ )
<b>Line</b> [{ $\{x_1, y_1\}, \dots, \{x_n, y_n\}$ }]	lomljena črta skozi točke ( $x_1, y_1$ ), ..., ( $x_n, y_n$ )
<b>Circle</b> [{ $x$ , $y$ ], $r$ ]	krožnica s središčem ( $x$ , $y$ ) in polmerom $r$
<b>Circle</b> [{ $x$ , $y$ ], $r_1$ , $r_2$ ]	elipsa s središčem ( $x$ , $y$ ) in polmeroma $r_1$ in $r_2$
<b>Disk</b> [{ $x$ , $y$ ], $r$ ]	krog s središčem ( $x$ , $y$ ) in polmerom $r$
<b>Disk</b> [{ $x$ , $y$ ], $r_1$ , $r_2$ ]	polna elipsa s središčem ( $x$ , $y$ ) in polmeroma $r_1$ in $r_2$
<b>Polygon</b> [{ $\{x_1, y_1\}, \dots, \{x_n, y_n\}$ }]	poligon z oglišči ( $x_1, y_1$ ), ..., ( $x_n, y_n$ )
<b>Rectangle</b> [{ $x_1, y_1$ ], [ $x_2, y_2$ ]}]	pravokotnik z nasprotnima ogliščema ( $x_1, y_1$ ) in ( $x_2, y_2$ )
<b>Text</b> [ $t$ , { $x$ , $y$ }]	besedilo $t$ s koordinatami ( $x$ , $y$ )

Poleg osnovnih elementov imamo še elemente, s katerimi nadzorujemo barve, debeline črt ipd. Naštejmo nekaj takih elementov:

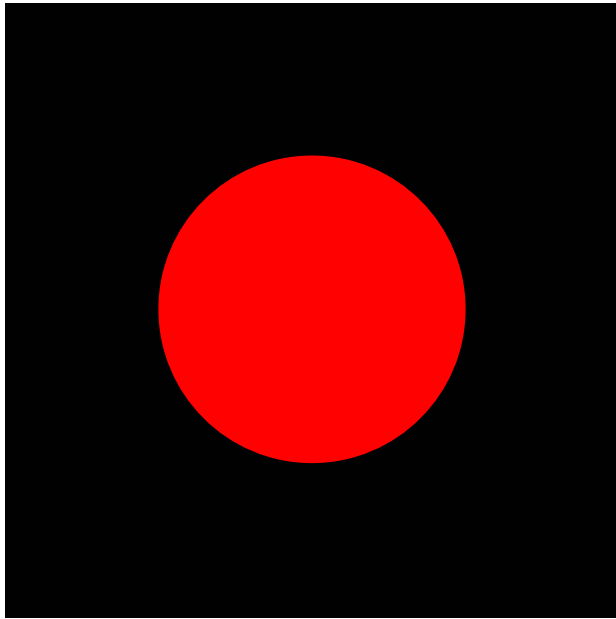
<b>RGBColor</b> [ $r$ , $g$ , $b$ ]	nastavi trenutno barvo na barvo z RGB komponentami $r$ , $g$ in $b$
<b>Hue</b> [ $h$ , $s$ , $v$ ]	nastavi trenutno barvo na barvo s HSV komponentami $h$ , $s$ in $v$
<b>Hue</b> [ $h$ ]	nastavi trenutno barvo na barvo $h$
<b>AbsolutePointSize</b> [ $r$ ]	nastavi premer točke na $d$
<b>PointSize</b> [ $r$ ]	nastavi premer točke na $r$ -ti delež širine slike
<b>AbsoluteThickness</b> [ $t$ ]	nastavi debelino črt na $t$
<b>Thickness</b> [ $r$ ]	nastavi debelino črte na $r$ -ti delež širine slike

Dvodimenzionalna slika je predstavljena z izrazom oblike

**Graphics** [{ $e_1, \dots, e_n$ }]

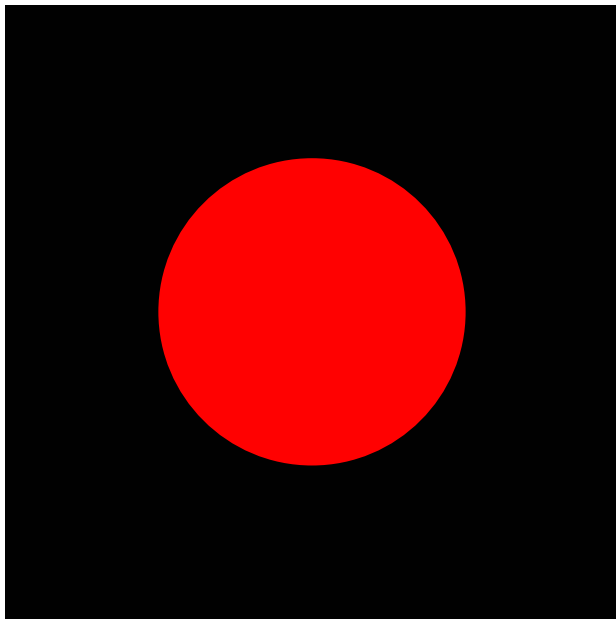
kjer so  $e_1, \dots, e_n$  grafični elementi, ki opisujejo sliko. Izrazi  $e_i$  so lahko tudi poljubno vgnezdjeni seznama, ki vsebujejo grafične elemente. Sliko prikažemo z ukazom **Show**. Narišimo sliko, ki vsebuje črn kvadrat in rdeč krog v kvadratu:

```
abstraktnaSlika =  
Graphics[{Rectangle[{-2, -2}, {2, 2}], RGBColor[1, 0, 0], Disk[{0, 0}, 1]}]
```



Slika je v verziji Mathematice 5.0 ali starejši sploščena, ker Mathematica vsako sliko raztegne tako, da je razmerje med širino in višino slike enako zlatemu rezu  $(1 + \sqrt{5})/2$ . Če tega ne želimo, uporabimo opcijo **AspectRatio** z vrednostjo **Automatic**.

```
Show[abstraktnaSlika, AspectRatio -> Automatic]
```



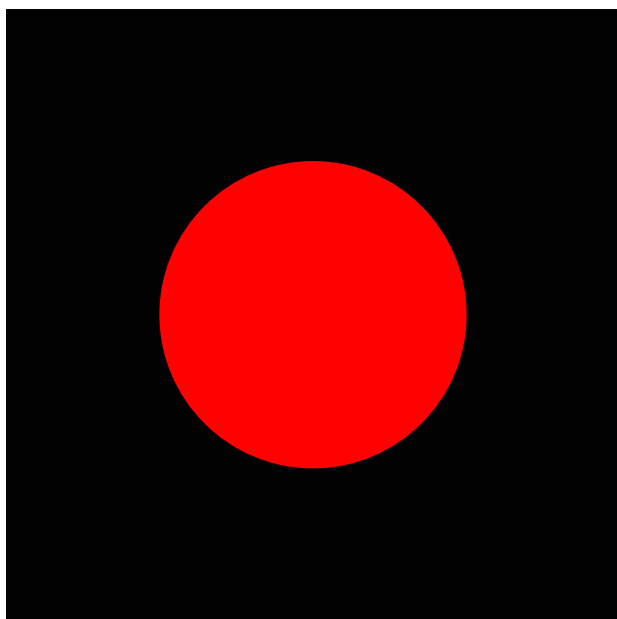
Če želimo nastaviti opcijo **AspectRatio** na **Automatic** za stalno, to storimo z ukazom **SetOptions**:

```
SetOptions [Graphics, AspectRatio → Automatic]
```

```
{AlignmentPoint → Center, AspectRatio → Automatic, Axes → False,  
AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,  
BaselinePosition → Automatic, BaseStyle → {}, ColorOutput → Automatic,  
ContentSelectable → Automatic, DisplayFunction → $DisplayFunction, Epilog → {},  
FormatType → TraditionalForm, Frame → False, FrameLabel → None, FrameStyle → {},  
FrameTicks → Automatic, FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {},  
ImageMargins → 0., ImagePadding → All, ImageSize → Automatic, LabelStyle → {},  
Method → Automatic, PlotLabel → None, PlotRange → All, PlotRangeClipping → False,  
PlotRangePadding → Automatic, PlotRegion → Automatic, PreserveImageOptions → Automatic,  
Prolog → {}, RotateLabel → True, Ticks → Automatic, TicksStyle → {}}
```

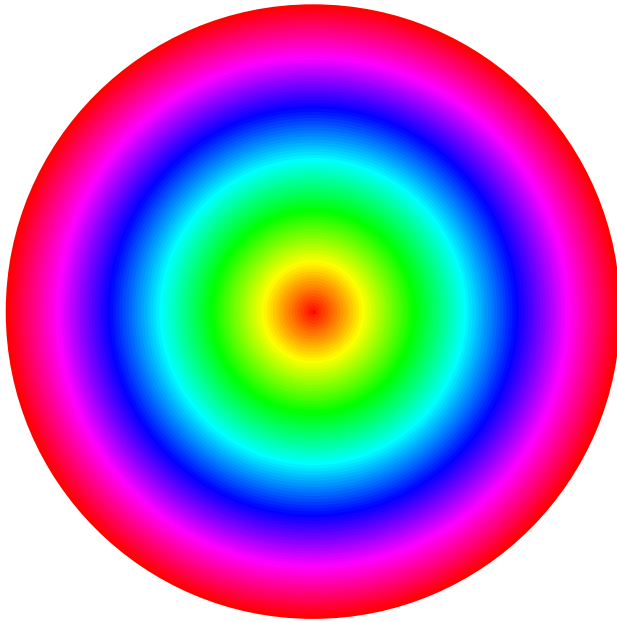
Sedaj se kvadrat zares nariše kot kvadrat, tudi če ne uporabimo **AspectRatio→Automatic**:

```
Show [abstraktnaSlika ]
```



Pa še ena malo bolj kičasta slika:

```
psiho = Graphics [Table[{Hue[h], Disk[{0, 0}, h]}, {h, 1, 0, -0.01}]]
```



Slika lahko z ukazom **Export** shranimo v datoteko, v enem od mnogih formatov (GIF, JPG, Encapsulated Postscript, PDF,...):

?? **Export**

`Export["file.ext", expr]` exports data to a file, converting it to the format corresponding to the file extension *ext*.  
`Export[file, expr, "format"]` exports data in the specified format.  
`Export[file, exprs, elems]` exports data by treating *exprs* as elements specified by *elems*. >>

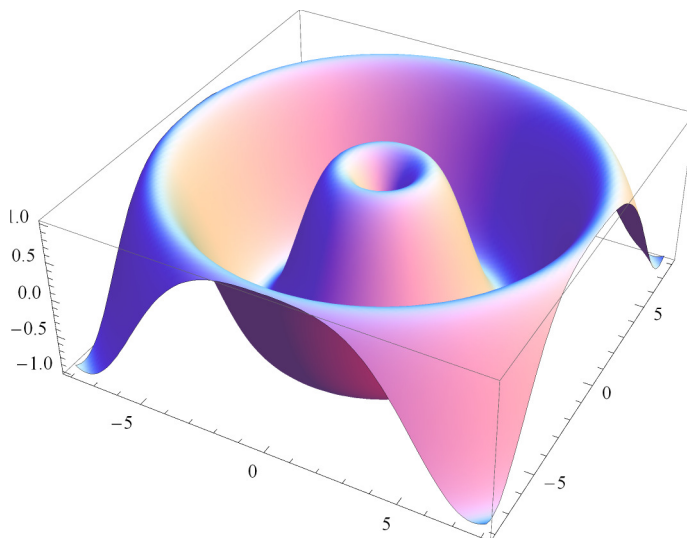
`Attributes[Export] = {Protected, ReadProtected}`

```
Export["psiho.pdf", psiho]
```

```
psiho.pdf
```

Še en primer uporabe ukaza **Export**:

```
pic = Plot3D[Sin[Sqrt[x^2 + y^2]], {x, -8, 8}, {y, -8, 8}, Mesh -> False, PlotPoints -> 100]
```



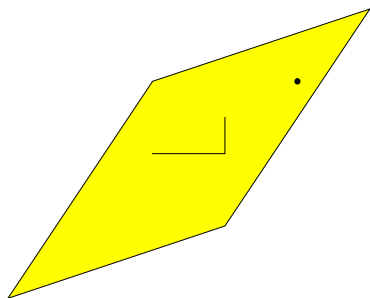
```
Export ["graf3d.eps", pic, "EPS"]
```

```
$Aborted
```

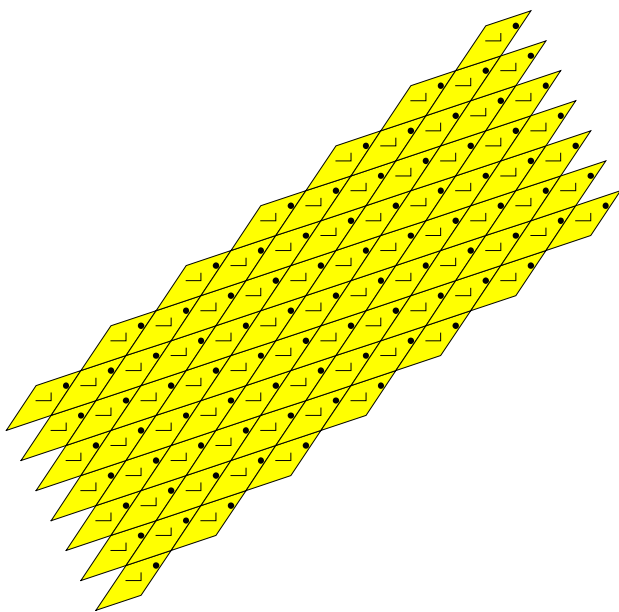
## Tlakovanja ravnine

### Tlakovanje s ploščico

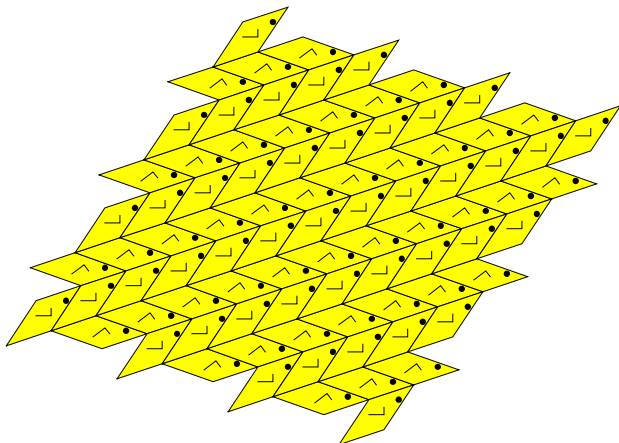
Naj bo  $p$  ravninski lik, imenovan tudi *ploščica*, s katerim želimo tlakovati ravnino. *Tlakovanje ravnine* dobimo tako, da celotno ravnino prekrijemo s kopijami  $p$  tako, da se ploščice ne prekrivajo. Seveda ravnine ne moremo tlakovati s ploščico poljubne oblike. S paralelogramsko ploščico



ravnino lahko tlakujemo takole:



Tlakovanje dobimo tako, da začetno ploščico premikamo vzdolž ene in druge stranice paralelograma. To pa ni edini način tlakovanja. Tu je še eden:



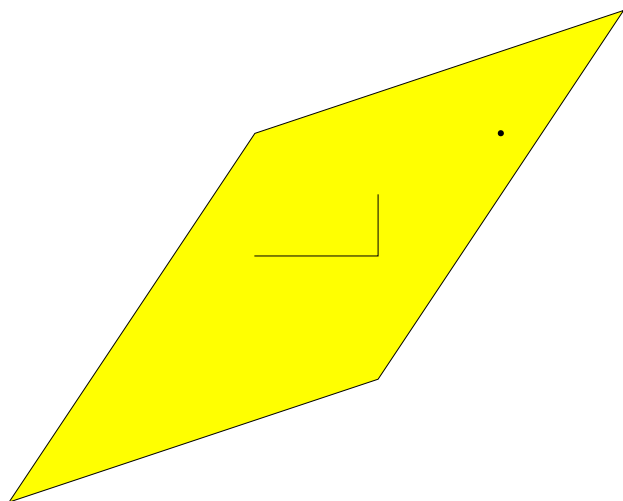
V tem primeru smo začetno ploščico bodisi premaknili vzdolž daljše stranice ali pa smo jo preko nje prezrcalili.

Osnovna ploščica je paralelogram določen s smernima vektorjema  $(3, 1)$  in  $(2, 3)$ . Ploščici dodamo "plavut" in "oko", da bomo lahko ločili različne orientacije:

```
ploscica = {RGBColor[1, 1, 0],
  Polygon[{{0, 0}, {3, 1}, {5, 4}, {2, 3}}],
  RGBColor[0, 0, 0],
  Line[{{0, 0}, {3, 1}, {5, 4}, {2, 3}, {0, 0}}],
  Line[{{3, 5/2}, {3, 2}, {2, 2}}],
  AbsolutePointSize[3],
  Point[{4, 3}]
}

{RGBColor[1, 1, 0], Polygon[{{0, 0}, {3, 1}, {5, 4}, {2, 3}}],
  RGBColor[0, 0, 0], Line[{{0, 0}, {3, 1}, {5, 4}, {2, 3}, {0, 0}}],
  Line[{{3, 5/2}, {3, 2}, {2, 2}}], AbsolutePointSize[3], Point[{4, 3}]}

Show[Graphics[ploscica]]
```



### Izometrije in afine transformacije

Tlakovanje generiramo tako, da osnovno ploščico preslikamo z izometrijami ravnine. Spomnimo se, da je *izometrija* taka preslikava ravnine, ki ohranja razdalje. Izometrija je *afina* preslikava, kar pomeni, da slika premice v premice.

Vsaka afina transformacija ravnine je podana z  $2 \times 2$  matriko  $a$  in vektorjem  $b$ , ki določata preslikavo

$$x \mapsto a \cdot x + b$$

V Mathematici afino transformacijo predstavimo s funkcijo `afina[a,b]`:

```
ClearAll [afina]
afina [a_, b_] [x : {_, _}] := a.x + b
```

Uporabili smo vzorec `{_, _}`, ki ustreza dvodimenzionalnemu vektorju, in ga poimenovali **x**.

Kompozicija afine preslikave `afina[a, b]` in `afina[c, d]` je afina preslikava `afina[a.c, a.d + b]`:

```
ClearAll [kompozicija]
kompozicija [afina [a_, b_], afina [c_, d_]] := afina [a.c, a.d + b]

kompozicija [u_afina, v_afina, w_++] := kompozicija [kompozicija [u, v], w]
```

Ker želimo z afinimi transformacijami slikati celotne slike, dodamo še pravila, ki ustrezno preslikajo grafične elemente. Omejimo se na preslikave točk, daljic in poligonov:

```
afina [a_, b_] [Point [x_]] := Point [afina [a, b] [x]]
afina [a_, b_] [Line [xs_]] := Line [afina [a, b] /@ xs]
afina [a_, b_] [Polygon [xs_]] := Polygon [afina [a, b] /@ xs]
afina [a_, b_] [c_] := c
```

Zadnji primer je za preslikavanje elementov, kot so `RGBColor` ali `PointSize`, ki jih afina transformacija ne spremeni (pravzaprav bi lahko spremenili velikost točke, a tega ne bomo počeli).

Osnove izometrije so translacija, rotacija in zrcaljenje preko premice. Definirajmo jih kot afine preslikave. Najprej translacija za smerni vektor  $(x, y)$ :

```
ClearAll [translacija]
translacija [x_, y_] := afina [{{1, 0}, {0, 1}}, {x, y}]
```

Rotacija `rotacija[φ]` je rotacija za kot  $\phi$  s središčem v izhodišču. Rotacija `rotacija[φ,x,y]` je rotacija za kot  $\phi$  s središčem v točki  $(x, y)$ .

```
ClearAll [rotacija]
rotacija [φ_] := afina [{{Cos [φ], -Sin [φ]}, {Sin [φ], Cos [φ]}}, {0, 0}]
rotacija [φ_, x_, y_] :=
  kompozicija [translacija [x, y], rotacija [φ], translacija [-x, -y]]
```

Zrcaljenje lahko podamo na tri načine: `zrcaljenje[φ]` je zrcaljenje preko premice skozi izhodišče z naklonom  $\phi$ ; `zrcaljenje[φ,x,y]` je zrcaljenje preko premice skozi točko  $(x, y)$  z naklonom  $\phi$ ; `zrcaljenje[x,y,u,v]` je zrcaljenje preko premice določene s točkama  $(x, y)$  in  $(u, v)$ .

```
ClearAll [zrcaljenje]

zrcaljenje [φ_] := afina [{{Cos [2 φ], Sin [2 φ]}, {Sin [2 φ], -Cos [2 φ]}}, {0, 0}]

zrcaljenje [φ_, x_, y_] :=
  kompozicija [translacija [x, y], zrcaljenje [φ], translacija [-x, -y]]

zrcaljenje [x_, y_, u_, v_] :=
  kompozicija [translacija [x, y], zrcaljenje [ArcTan [u - x, v - y]], translacija [-x, -y]]
```

### Kako generiramo tlakovanje

Tlakovanje ravnine je določeno s ploščico  $p$  in osnovnimi izometrijami  $f_1, \dots, f_n$ , s katerimi so določeni osnovni premiki ploščice. Vsako drugo ploščico tlakovanja dobimo z zaporedno uporabo osnovnih premikov:

$$f_{i_1} [f_{i_2} [\dots f_{i_k} [p] \dots]]$$

Na primer, z osnovnima translacijama

```

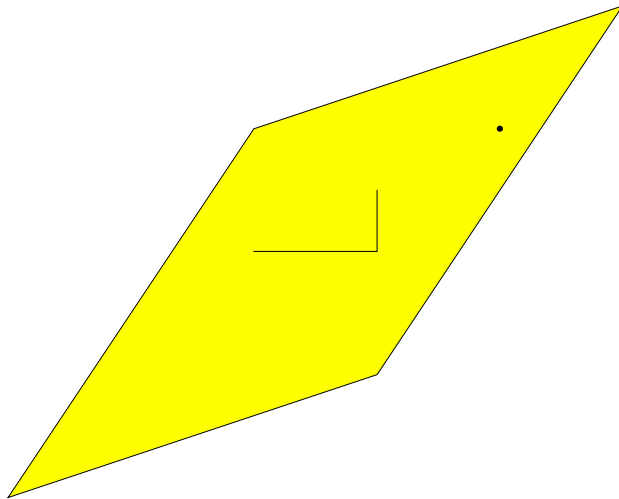
t1 = translacija[3, 1]
t2 = translacija[2, 3]

afina[{{1, 0}, {0, 1}}, {3, 1}]

afina[{{1, 0}, {0, 1}}, {2, 3}]

```

in ploščico



lahko tlakujemo ravnino tako, da uporabimo razne kombinacije translacij **t1** in **t2** na **p**. Ker je **p seznam** grafičnih elementov, je treba pisati **t1/@p** namesto **t1[p]**.

**ploscica**

```

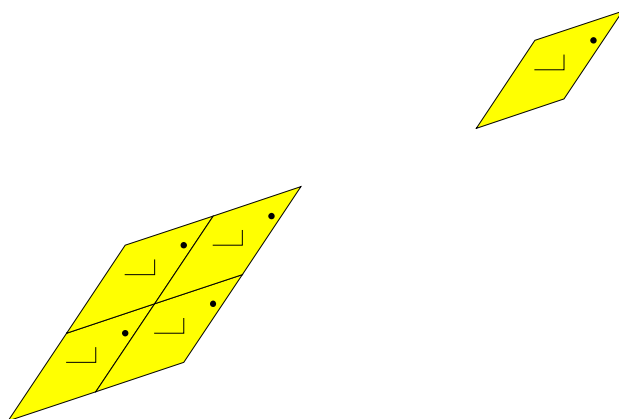
{RGBColor[1, 1, 0], Polygon[{{0, 0}, {3, 1}, {5, 4}, {2, 3}]},
 RGBColor[0, 0, 0], Line[{{0, 0}, {3, 1}, {5, 4}, {2, 3}, {0, 0}}],
 Line[{{3, 5/2}, {3, 2}, {2, 2}}], AbsolutePointSize[3], Point[{{4, 3}}]

```

```

Graphics[{plocscica, t1/@plocscica, t2/@plocscica,
  kompozicija[t1, t2] /@plocscica, kompozicija[t1, t1, t1, t2, t1, t2] /@plocscica}] // Show

```



Napišimo funkcijo, ki sistematično generira kombinacije izometrij. Najprej bomo potrebovali funkcijo, ki sprejme funkcijo **f** in seznama  $\{x_1, x_2, \dots, x_n\}$  in  $\{y_1, y_2, \dots, y_k\}$  in napravi seznam elementov oblike  $f[x_i, y_j]$ . Za ta namen je uporabna vgrajena funkcija **Outer**:

**Outer[f, {x1, x2, x3}, {y1, y2, y3, y4}]**

```

{{f[x1, y1], f[x1, y2], f[x1, y3], f[x1, y4]},
 {f[x2, y1], f[x2, y2], f[x2, y3], f[x2, y4]}, {f[x3, y1], f[x3, y2], f[x3, y3], f[x3, y4]}}

```

Ker **Outer** vrne matriko (seznam seznamov) uporabimo še funkcijo **Flatten**, ki seznam seznamov "splošči":

```
Flatten[Outer[f, {x1, x2, x3}, {y1, y2, y3, y4}]]
{f[x1, y1], f[x1, y2], f[x1, y3], f[x1, y4], f[x2, y1], f[x2, y2],
 f[x2, y3], f[x2, y4], f[x3, y1], f[x3, y2], f[x3, y3], f[x3, y4]}
```

**Flatten** in **Outer** delujeta na seznamih seznamov seznamov..., kolikor so pač vgnezdjeni. Če hočemo, da delujeta samo na seznamih seznamov, jima podamo dodatni argument, ki določa globino vgnezdenja, do katere naj delujeta:

```
Flatten[{{1, 2}, {{3}}, {4, 5}}, {6}]
{1, 2, 3, 4, 5, 6}

Flatten[{{1, 2}, {{3}}, {4, 5}}, {6}], 1]
{1, 2, {{3}}, {4, 5}, 6}
```

Napišimo funkcijo **flatOuter**, ki sprejme funkcijo **f**, seznama **x** in **y** ter vrne seznam vseh kombinacij  $f[u, v]$ , kjer sta **u** in **v** elementa **x** in **y**:

```
ClearAll[flatOuter]
flatOuter[f_, x_, y_] := Union@Flatten[Outer[f, x, y, 1], 1]
```

Dodali smo še **Union**, da se znebimo elementov, ki se ponavljajo. Primer:

```
flatOuter[Plus, {a, b, c}, {a, b, c}]
{2 a, 2 b, a + b, 2 c, a + c, b + c}

flatOuter[f, {x, y, z}, {s, t}]
{f[x, s], f[x, t], f[y, s], f[y, t], f[z, s], f[z, t]}
```

Funkcija **generate[f, x, n]** zgenerira vse kombinacije izraze globine **n**, ki jih dobi tako, da funkcijo **f** uporabi na elementih seznama **x** in izrazih globine **n**. Pri tem **f** sprejme dva argumenta:

```
ClearAll[generate]
generate[f_, x_List, n_Integer] :=
  Union@Flatten[NestList[Function[y, flatOuter[f, x, y]], x, n], 1]
```

Uporabili smo funkcijo **NestList**:

```
Nest[f, x, 5]
f[f[f[f[f[x]]]]]

NestList[f, x, 5]
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]], f[f[f[f[f[x]]]]]}

generate[f, {a, b, c}, 0]
{a, b, c}

generate[f, {a, b, c}, 1]
{a, b, c, f[a, a], f[a, b], f[a, c], f[b, a], f[b, b], f[b, c], f[c, a], f[c, b], f[c, c]}
```

```
generate[f, {a, b, c}, 3]
```

```
{a, b, c, f[a, a], f[a, b], f[a, c], f[a, f[a, a]], f[a, f[a, b]], f[a, f[a, c]],
f[a, f[a, f[a, a]]], f[a, f[a, f[a, b]]], f[a, f[a, f[a, c]]], f[a, f[a, f[b, a]]],
f[a, f[a, f[b, b]]], f[a, f[a, f[b, c]]], f[a, f[a, f[c, a]]], f[a, f[a, f[c, b]]],
f[a, f[a, f[c, c]]], f[a, f[b, a]], f[a, f[b, b]], f[a, f[b, c]], f[a, f[b, f[a, a]]],
f[a, f[b, f[a, b]]], f[a, f[b, f[a, c]]], f[a, f[b, f[b, a]]], f[a, f[b, f[b, b]]],
f[a, f[b, f[b, c]]], f[a, f[b, f[c, a]]], f[a, f[b, f[c, b]]], f[a, f[b, f[c, c]]],
f[a, f[c, a]], f[a, f[c, b]], f[a, f[c, c]], f[a, f[c, f[a, a]]], f[a, f[c, f[a, b]]],
f[a, f[c, f[a, c]]], f[a, f[c, f[b, a]]], f[a, f[c, f[b, b]]], f[a, f[c, f[b, c]]],
f[a, f[c, f[c, a]]], f[a, f[c, f[c, b]]], f[a, f[c, f[c, c]]], f[b, a], f[b, b], f[b, c],
f[b, f[a, a]], f[b, f[a, b]], f[b, f[a, c]], f[b, f[a, f[a, a]]], f[b, f[a, f[a, b]]],
f[b, f[a, f[a, c]]], f[b, f[a, f[b, a]]], f[b, f[a, f[b, b]]], f[b, f[a, f[b, c]]],
f[b, f[a, f[c, a]]], f[b, f[a, f[c, b]]], f[b, f[a, f[c, c]]], f[b, f[b, a]], f[b, f[b, b]],
f[b, f[b, c]], f[b, f[b, f[a, a]]], f[b, f[b, f[a, b]]], f[b, f[b, f[a, c]]],
f[b, f[b, f[b, a]]], f[b, f[b, f[b, b]]], f[b, f[b, f[b, c]]], f[b, f[b, f[c, a]]],
f[b, f[b, f[c, b]]], f[b, f[b, f[c, c]]], f[b, f[c, a]], f[b, f[c, b]], f[b, f[c, c]],
f[b, f[c, f[a, a]]], f[b, f[c, f[a, b]]], f[b, f[c, f[a, c]]], f[b, f[c, f[b, a]]],
f[b, f[c, f[b, b]]], f[b, f[c, f[b, c]]], f[b, f[c, f[c, a]]], f[b, f[c, f[c, b]]],
f[b, f[c, f[c, c]]], f[c, a], f[c, b], f[c, c], f[c, f[a, a]], f[c, f[a, b]],
f[c, f[a, c]], f[c, f[a, f[a, a]]], f[c, f[a, f[a, b]]], f[c, f[a, f[a, c]]],
f[c, f[a, f[b, a]]], f[c, f[a, f[b, b]]], f[c, f[a, f[b, c]]], f[c, f[a, f[c, a]]],
f[c, f[a, f[c, b]]], f[c, f[a, f[c, c]]], f[c, f[b, a]], f[c, f[b, b]], f[c, f[b, c]],
f[c, f[b, f[a, a]]], f[c, f[b, f[a, b]]], f[c, f[b, f[a, c]]], f[c, f[b, f[b, a]]],
f[c, f[b, f[b, b]]], f[c, f[b, f[b, c]]], f[c, f[b, f[c, a]]], f[c, f[b, f[c, b]]],
f[c, f[b, f[c, c]]], f[c, f[c, a]], f[c, f[c, b]], f[c, f[c, c]], f[c, f[c, f[a, a]]],
f[c, f[c, f[a, b]]], f[c, f[c, f[a, c]]], f[c, f[c, f[b, a]]], f[c, f[c, f[b, b]]],
f[c, f[c, f[b, c]]], f[c, f[c, f[c, a]]], f[c, f[c, f[c, b]]], f[c, f[c, f[c, c]]]
```

Sedaj lahko generiramo translacije paralelograma:

```
generate[kompozicija, {translacija[3, 1], translacija[2, 3]}, 2]
```

```
{afina[{{1, 0}, {0, 1}}, {2, 3}],
afina[{{1, 0}, {0, 1}}, {3, 1}], afina[{{1, 0}, {0, 1}}, {4, 6}],
afina[{{1, 0}, {0, 1}}, {5, 4}], afina[{{1, 0}, {0, 1}}, {6, 2}],
afina[{{1, 0}, {0, 1}}, {6, 9}], afina[{{1, 0}, {0, 1}}, {7, 7}],
afina[{{1, 0}, {0, 1}}, {8, 5}], afina[{{1, 0}, {0, 1}}, {9, 3}]}
```

Tlakovanje dobimo tako, da seznam afinih preslikav, ki jih generiramo z **generate**, uporabimo na osnovni ploščici **p**:

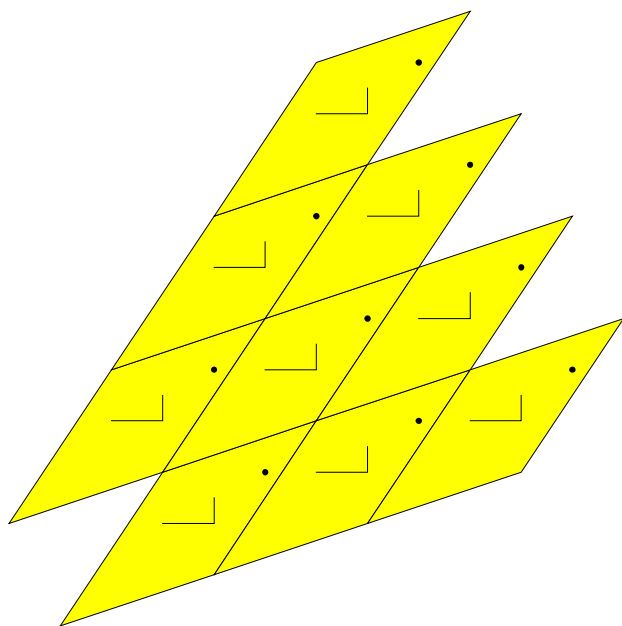
```
ClearAll[tlakovanje]
```

```
tlakovanje[op_, gen_, p_, n_Integer] :=
```

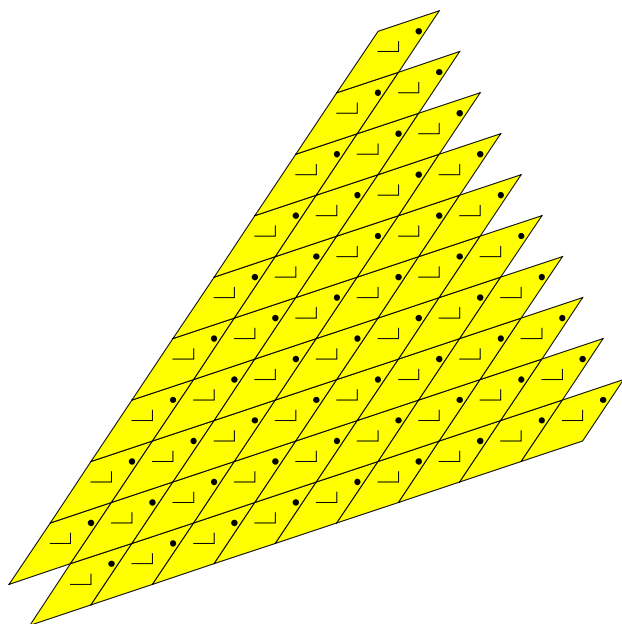
```
Graphics[Function[f, f /@ p] /@ generate[op, gen, n]]
```

Napravimo tlakovanje s translacijama:

```
tlakovanje[kompozicija, {translacija[3, 1], translacija[2, 3]}, ploscica, 2] // Show
```

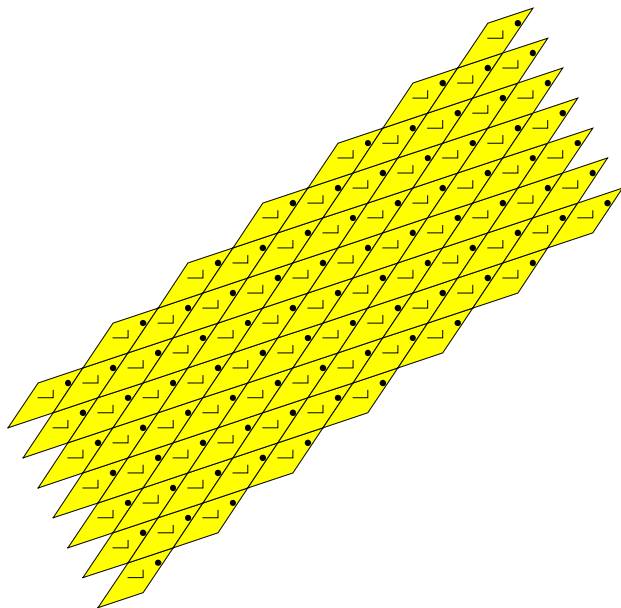


```
tlakovanje[kompozicija, {translacija[3, 1], translacija[2, 3]}, ploscica, 8] // Show
```



Da pokrijemo celo ravnino, moramo premikati ploščico še v smereh  $(-3, -1)$  in  $(-2, -3)$ :

```
tlakovanje[kompozicija, {translacija[3, 1], translacija[2, 3],
  translacija[-2, -3], translacija[-3, -1]}, ploscica, 5] // Show
```



Kaj pa tlakovanje z zrcaljenjem?

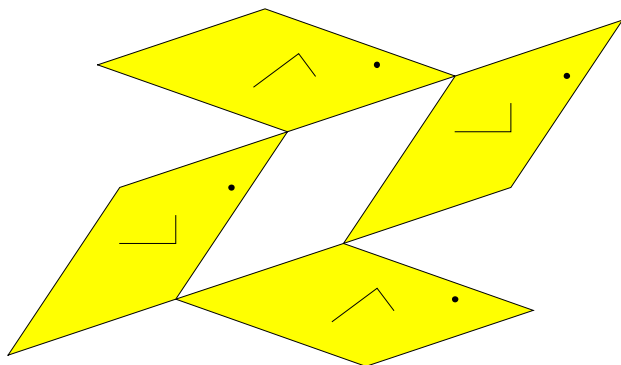
```
p1 = (zrcaljenje[0, 0, 3, 1] // FullSimplify)
```

```
afina[{{{\frac{4}{5}, \frac{3}{5}}, {\frac{3}{5}, -\frac{4}{5}}}, {0, 0}}
```

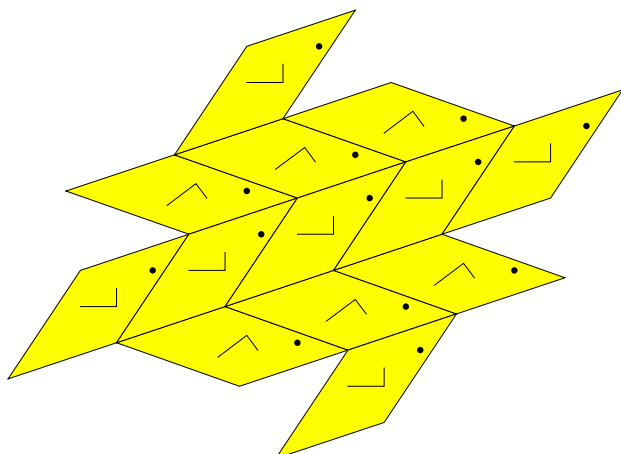
```
p2 = (zrcaljenje[2, 3, 5, 4] // FullSimplify)
```

```
afina[{{{\frac{4}{5}, \frac{3}{5}}, {\frac{3}{5}, -\frac{4}{5}}}, {-\frac{7}{5}, \frac{21}{5}}}}
```

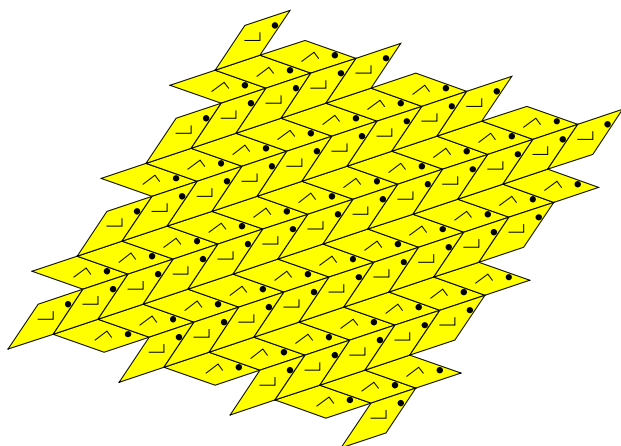
```
tlakovanje[kompozicija,
  {p1, p2, translacija[3, 1], translacija[-3, -1]}, ploscica, 0] // Show
```



```
tlakovanje[kompozicija,
  {p1, p2, translacija[3, 1], translacija[-3, -1]}, ploscica, 1] // Show
```



```
tlakovanje[kompozicija,
  {p1, p2, translacija[3, 1], translacija[-3, -1]}, ploscica, 5] // Show
```



### Naloga: tlakovanje s šestkotniki

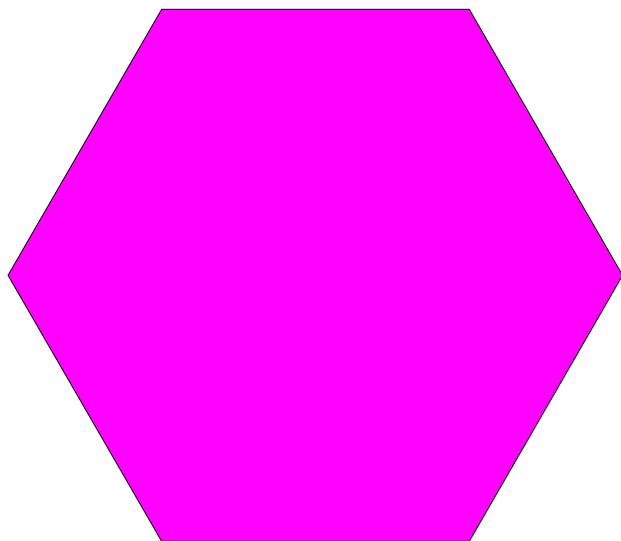
Čebele znajo tlakovati ravnino s šestkotniki.

```
ClearAll[polarne]
polarne[φ_, r_] := {r * Cos[φ], r * Sin[φ]}

sestkotnik = {
  RGBColor[1, 0, 1],
  Polygon[Table[polarne[k * Pi / 3, 1], {k, 0, 5}]],
  RGBColor[0, 0, 0],
  Line[Table[polarne[k * Pi / 3, 1], {k, 0, 6}]]
}

{RGBColor[1, 0, 1],
 Polygon[{{1, 0}, {1/2, sqrt(3)/2}, {-1/2, sqrt(3)/2}, {-1, 0}, {-1/2, -sqrt(3)/2}, {1/2, -sqrt(3)/2}}],
 RGBColor[0, 0, 0],
 Line[{{1, 0}, {1/2, sqrt(3)/2}, {-1/2, sqrt(3)/2}, {-1, 0}, {-1/2, -sqrt(3)/2}, {1/2, -sqrt(3)/2}, {1, 0}}]}
```

```
Show[Graphics[sestkotnik]]
```



Ravnino lahko tlakujemo s šestkotniki tako, da šestotnik rotiramo okrog enega oglišča za kot  $2\pi/3$  in okrog sosednjega oglišča za kot  $2\pi/3$ . Se pravi, da je tlakovanje generirano z dvema rotacijama. Poskusi!