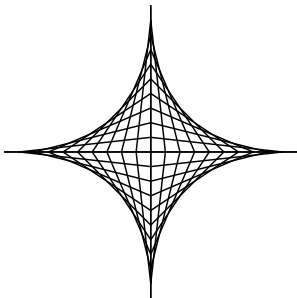
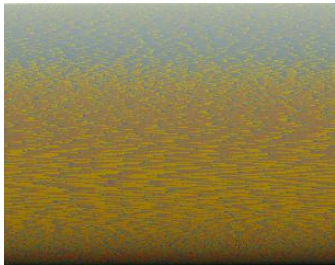
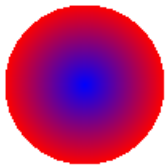


# Obrázky (reprezentace, generování, úpravy)

IB111 Úvod do programování

2016



- procvičení základních konstrukcí z jiného pohledu
- propojení programování a matematiky
- téma „reprezentace dat“
- procvičení „čtení kódu“
- podklad pro zajímavé cvičení

# Poznámka k efektivitě, obrázkům

ukázky programů v přednášce:

- snaha o čitelnost programů
- neefektivní (pomalé):
  - algoritmy
  - technická realizace (např. putpixel vs load + pixel access object)

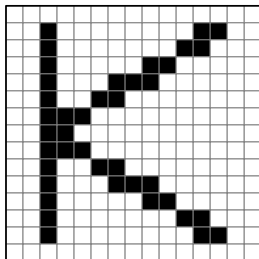
nízká / rozličná kvalita obrázků – čistě pragmatické důvody (nepříliš velké PDF), žádná skrytá pointa

obrázky, zvuk, video:

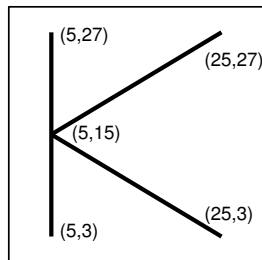
- kniha *Introduction to Computing and Programming in Python, A Multimedia Approach*, M. J. Guzdial, B. Ericson.
- <http://coweb.cc.gatech.edu/mediaComp-teach>

# Reprezentace obrázků

Bitmapová grafika

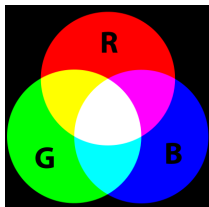


Vektorová grafika



# Reprezentace barev

- více barevných modelů (aditivní, subtraktivní)
- budeme používat aditivní model RGB – red, green, blue
- každá složka = hodnota 0-255 (8 bitů, 1 byte)
- barva = trojice, např. (15, 255, 100)



# Knihovny (moduly) v Pythonu

- knihovna poskytuje rozšiřující funkcionalitu
- zdroje knihoven:
  - standardní distribuce (např. math, turtle)
  - separátní instalace (např. numpy)
  - vlastní knihovny
- použití knihoven:
  - `import knihovna` – následná volání `knihovna.funkce()`
  - `import knihovna as nazev`
  - `from knihovna import funkce`
  - `from knihovna import *` (nedoporučeno)



# Proč nepoužívat „import \*“

```
from X import *
```

V malém programu nemusí vadit, ale ve větších projektech považováno za velmi špatnou praxi.

- „nepořádek“ v jmenném prostoru
- kolize, přepis
- překlepy se mohou chovat magicky
- složitější interpretace chybových hlášek
- Python Zen: Explicit is better than implicit.

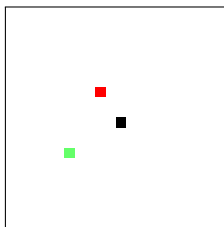
- knihovna pro práci s bitmapovými obrázky
- velmi bohatá funkcionalita
- použijeme jen základní operace:
  - `new` – vytvoření obrázku
  - `open`, `convert` – otevření obrázku, konverze na RGB mód
  - `getpixel` – zjištění barvy bodu
  - `putpixel` – změna barvy bodu
  - `size` – velikost obrázku
  - `show`, `save` – zobrazení, uložení

- Python Imaging Library (PIL): jen pro Python 2  
<http://www.pythonware.com/products/pil/>
- implementace Pillow (i pro Python 3):  
<https://pypi.python.org/pypi/Pillow/2.1.0>
- `from PIL import Image`

- reprezentace souřadnic a barev pomocí n-tic (tuple)
- podobné jako seznamy, ale neměnitelné; zápis pomocí kulatých závorek
- u obrázků typicky:
  - souřadnice:  $(x, y)$
  - barva:  $(r, g, b)$

# Image demo

```
def demo():  
    im = Image.new("RGB", (20,20), (255,255,255))  
        # model, velikost, barva pozadi  
    im.putpixel((10,10), (0,0,0))  
    im.putpixel((8,7), (255,0,0))  
    im.putpixel((5,13), (100,255,105))  
    im.show()  
    im.save("demo.png")
```



# Geometrické útvary

Napište programy pro generování následujících útvarů:

čtverec



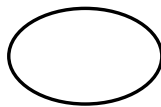
trojúhelník



kruh



elipsa



spirála

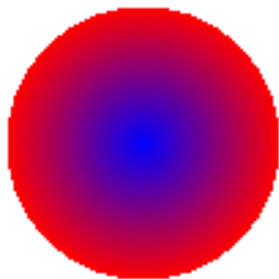


```
def disc(a = 150, r = 50):  
    im = Image.new("RGB", (a, a), (255, 255, 255))  
    for x in range(a):  
        for y in range(a):  
            if XXX:  
                im.putpixel((x, y), (0, 0, 0))  
    im.show()
```

```
def disc(a = 150, r = 50):  
    im = Image.new("RGB", (a, a), (255, 255, 255))  
    for x in range(a):  
        for y in range(a):  
            if (x-a/2)**2 + (y-a/2)**2 < r**2:  
                im.putpixel((x, y), (0, 0, 0))  
    im.show()
```



# Barevný kruh

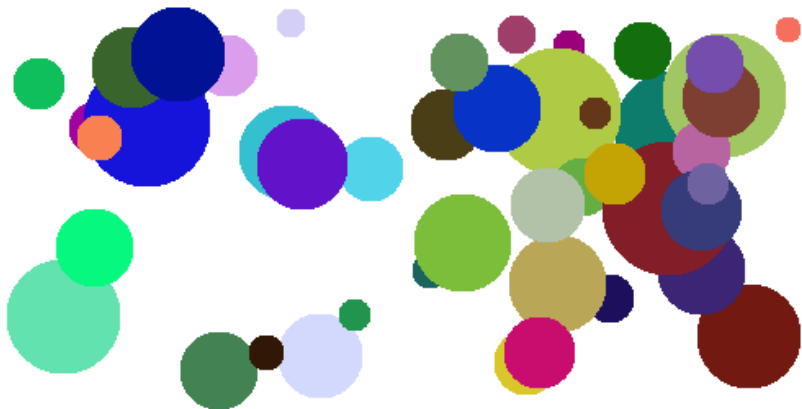


# Barevný kruh

Barvu „namícháme“ podle vzdálenosti od středu kruhu:

```
d = math.sqrt((x-a/2)**2 + (y-a/2)**2)
if d < r:
    c = int(255*d/r)
    im.putpixel((x,y), (c,0,255-c))
```

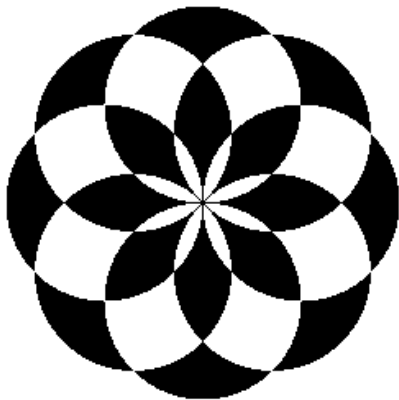
# Barevné kruhy



## Přidání náhodného kruhu do obrázku

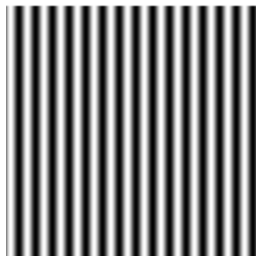
```
def add_random_disc(im):
    (width, height) = im.size
    r = random.randint(8, min(width, height) // 6)
    sx = random.randint(r+1, width-r-1)
    sy = random.randint(r+1, height-r-1)
    color = (random.randint(0,255),
             random.randint(0,255),
             random.randint(0,255))
    for x in range(width):
        for y in range(height):
            if (x-sx)**2 + (y-sy)**2 < r**2:
                im.putpixel((x,y), color)
```

# Námět na procvičení

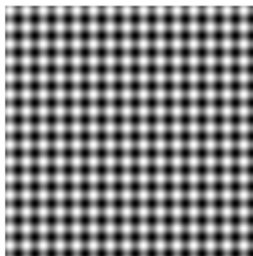


# Geometrické obrazce

pruhy



mřížka



vlny



# Základní princip

- potřebujeme plynulý přechod mezi bílou a černou
- jakou matematickou funkci využijeme?

# Základní princip

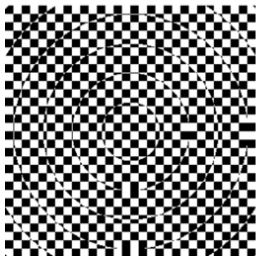
- potřebujeme plynulý přechod mezi bílou a černou
- jakou matematickou funkci využijeme?
- sinus – hodnoty mezi -1 a 1, perioda  $2\pi$
- potřebujeme – hodnoty mezi 0 a 255, perioda (např.) 20



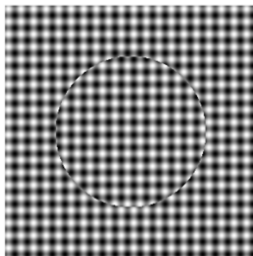
```
def strips(size = 150, count = 5):  
    im = Image.new("RGB", (size, size))  
    for x in range(size):  
        for y in range(size):  
            z = math.sin(count * 2 * math.pi * x/size)  
            shade = int(255 * (z + 1) / 2)  
            im.putpixel((x,y), (shade, shade, shade))  
    im.show()
```

# Vzory II

šachovnice a kruhy



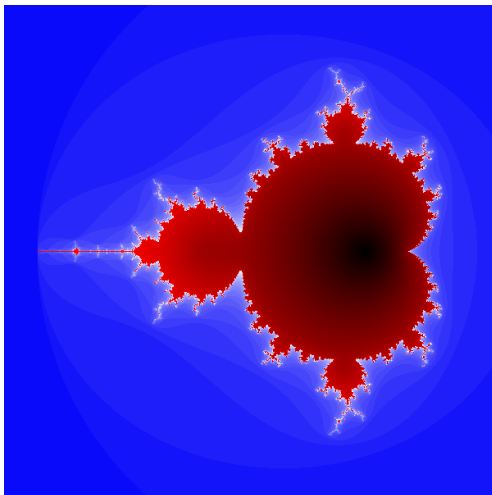
mřížka a kruh



vlny a čtverec



# Mandelbrotova množina

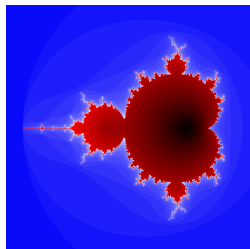


# Mandelbrotova množina

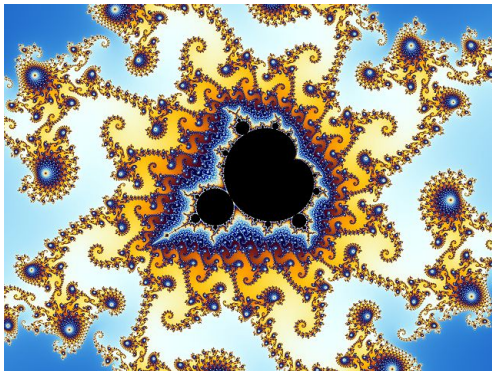
- $z_1 = 0$ ,  $c = x + yi$  je konstanta (komplexní číslo)
- definujeme posloupnost

$$z_{n+1} = z_n^2 + c$$

- $c$  patří do Mandelbrotovy množiny  $\Leftrightarrow$  tato posloupnost je omezená



# Mandelbrotova množina – detail



Zdroj: Wikipedia

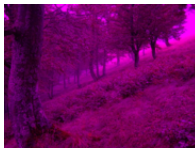
Video zoom: <http://www.youtube.com/watch?v=gEw8xp1aRA>

# Mandelbrotova množina – kód

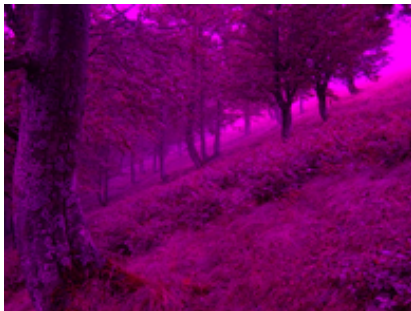
```
-
                                = (
                                255,
                                lambda
                                V, B, c
                                :c and Y(V*V+B,B, c
                                -1)if(abs(V)<6)else
                                2+c-4*abs(V)**-0.4)/i
                                (
                                ) ;v, x=1500,1000;C=range(v*x
                                );import struct;P=struct.pack;M,\
                                j ='<QIIHHHH',open('M.bmp','wb').write
                                for X in j('BM'+P(M,v*x*3+26,26,12,v,x,1,24))or C:
                                i ,Y=_;j(P('BBB',*(lambda T:(T*80+T**9
                                *i-950*T **99,T*70-880*T**18+701*
                                T **9 ,T*i**(1-T**45*2))))(sum(
                                [
                                Y(0,(A%3/3.+X%v+(X/v+
                                A/3/3.-x/2)/1j)*2.5
                                /x -2.7,i)**2 for \
                                A
                                in C
                                [:9]])
                                /9)
                                ) )
```

<http://preshing.com/20110926/high-resolution-mandelbrot-in-obfuscated-python/>

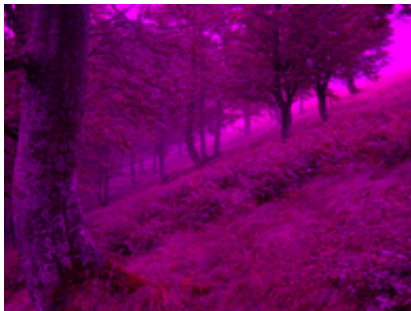
# Úpravy obrázků



# Úprava barev







pro každý pixel:

- zjistí barvu (getpixel)
- uloží upravenou barvu (putpixel)

# Úprava barev – kód

```
def remove_green(filename):  
    im = Image.open(filename)  
    im = im.convert("RGB")  
    width, height = im.size  
    for x in range(width):  
        for y in range(height):  
            (r, g, b) = im.getpixel((x, y))  
            im.putpixel((x, y), (r, 0, b))  
    im.show()
```

# Úprava barev – obecnější řešení

```
def transform_colors(filename, f_trans):
    im = Image.open(filename)
    im = im.convert("RGB")
    width, height = im.size
    for x in range(width):
        for y in range(height):
            (r,g,b) = im.getpixel((x,y))
            im.putpixel((x,y), f_trans(r,g,b))
    im.show()

def inversion(r, g, b):
    return (255-r, 255-g, 255-b)
transform_colors("les.jpg", inversion)
transform_colors("les.jpg", \
    lambda r, g, b: (255-r, b, g))
```

# Zrcadlový obraz



# Zrcadlový obraz



pro každý pixel v levé polovině:

- zjistí jeho barvu (getpixel)
- uloží barvu na příslušnou pozici v pravé polovině (putpixel)

## Zrcadlový obraz – kód

```
for x in range(width / 2):  
    for y in range(height):  
        im.putpixel((width-1-x,y),  
                    im.getpixel((x,y)))
```

# Překlopení



# Překlopení



prohazování symetrických bodů



V předchozím kódu (zrcadlový obraz) změňíme tělo for cyklu:

```
tmp = im.getpixel((width-1-x,y))
im.putpixel((width-1-x,y),
            im.getpixel((x,y)))
im.putpixel((x,y), tmp)
```

# Rotace



# Rotace

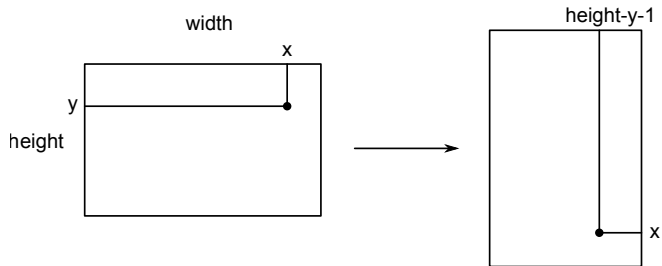


vytvoř nový obrázek a naplň jej pixely podle originálu – vhodně  
pozměněné souřadnice

# Rotace – kód s cenzurou

```
def rotation(filename):  
    im = Image.open(filename)  
    im = im.convert("RGB")  
    width, height = im.size  
    new_im = Image.new("RGB", (height, width))  
    for x in range(width):  
        for y in range(height):  
            new_im.putpixel((XXX, YYY),  
                             im.getpixel((x,y)))  
    new_im.show()
```

# Rotace – ilustrace



# Rotace o zadaný úhel



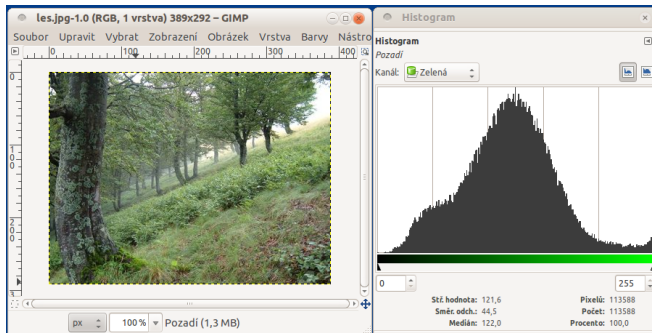
# Rotace o zadaný úhel



goniometrické funkce, lineární transformace, matice – aplikace  
(procvičení) pojmů z matematiky

# Histogram

variacie na téma „frekvenční analýza“





# Histogram – textový výpis

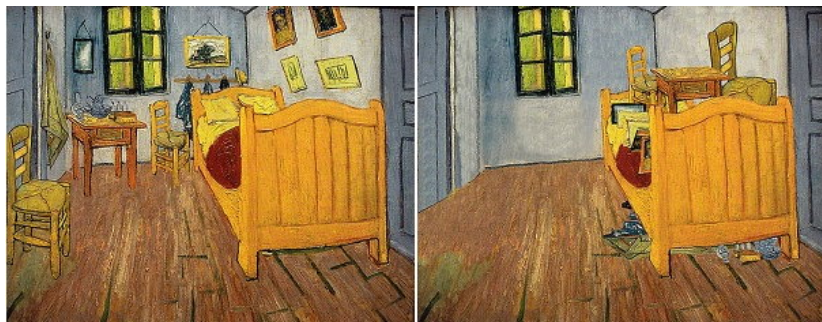
0 - 19:	0.3 %
20 - 39:	3.5 %
40 - 59:	6.3 %
60 - 79:	8.3 %
80 - 99:	12.7 %
100 - 119:	17.1 %
120 - 139:	18.5 %
140 - 159:	15.2 %
160 - 179:	9.0 %
180 - 199:	4.0 %
200 - 219:	1.8 %
220 - 239:	1.1 %
240 - 259:	2.2 %

(implementace – doporučené cvičení)

# Další náměty na úpravy

- změna velikosti obrázku
- převod do stupňů šedi
- rozmazání (blur), detekce hran
- ... další věci co umí váš grafický program

# Pořádek v umění

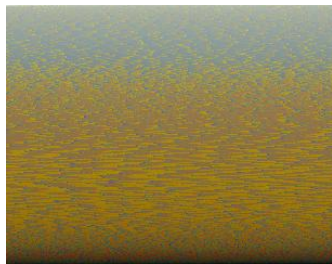


[http://www.ted.com/talks/ursus\\_wehrli\\_tidies\\_up\\_art.html](http://www.ted.com/talks/ursus_wehrli_tidies_up_art.html)

# Pořádek (nejen) v umění



# Pořádek v umění – pixel po pixelu

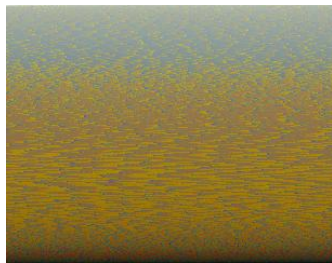


# Řazení pixelů podle barvy

- vytvoříme seznam všech použitých barev – seznam trojic  
[ (0, 150, 20), (255,255,255), (0, 0, 255),  
... ]
- seznam seřadíme
- barvy umístíme do obrázku

```
def tidy_up(filename):
    im = Image.open(filename)
    im = im.convert("RGB")
    width, height = im.size
    pixels = []
    for x in range(width):
        for y in range(height):
            pixels.append(im.getpixel((x, y)))
    pixels.sort()
    new_im = Image.new("RGB", (width, height))
    for y in range(height):
        for x in range(width):
            new_im.putpixel((x, y), pixels[y*width+x])
    new_im.show()
```

# Řazení pixelů





# Řazení pixelů

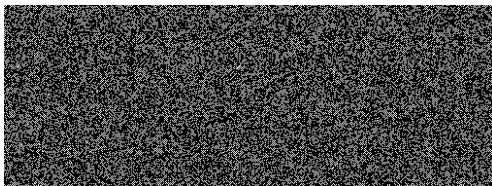
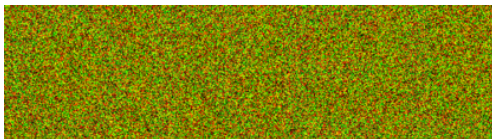
- pixels je seznam trojic (r, g, b)
- sort() používá „lexikografické“ řazení
- pokud chceme „řazení dle součtu“ (intenzity) nahradíme pixels.sort() za:

```
pixels = sorted(pixels, \
                key = lambda c: -(c[0]+c[1]+c[2]))
```

Zkuste další způsoby řazení:

- po řádcích / sloupcích
- po „čtverečcích“
- podle jiného kritéria
- „gradient“ po uhlopříčce

# Skrývačky v bitmapové grafice

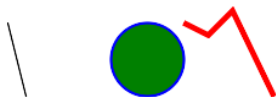


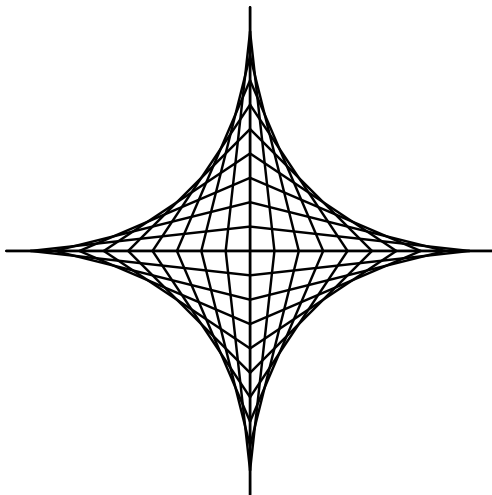
# Scalable Vector Graphics (SVG)

- vektorový formát založený na XML
- snadný způsob vytváření obrázků v jakémkoliv jazyce (generujeme prostý text)
- prohlížení: např. webový prohlížeč
- ruční editování: např. Inkscape
- převod na bitmapu: např. convert (ImageMagick)

# SVG příklad

```
<svg xmlns="http://www.w3.org/2000/svg">  
<line x1="15" y1="20" x2="30" y2="80"  
      stroke="black" stroke-width="1"/>  
<circle cx="130" cy="50" r="30" stroke="blue"  
        stroke-width="2" fill="green" />  
<polyline fill="none" stroke="red" stroke-width="4"  
          points="160,20 180,30 200,10 234,80"/>  
</svg>
```





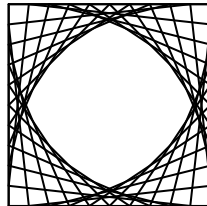
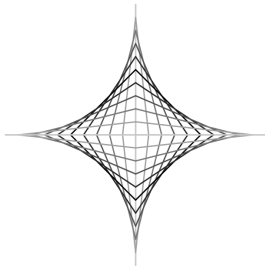
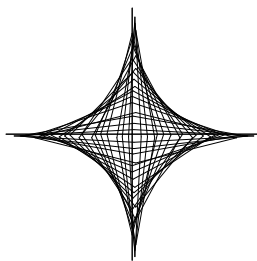
```
def star(n = 10, length = 100):
    svg_header()
    center_x = length * 1.5
    center_y = length * 1.5
    step = length / n
    for i in range(n + 1):
        svg_line(center_x + i*step, center_y,
                center_x, center_y + (n-i)*step)
        svg_line(center_x - i*step, center_y,
                center_x, center_y + (n-i)*step)
        svg_line(center_x + i*step, center_y,
                center_x, center_y - (n-i)*step)
        svg_line(center_x - i*step, center_y,
                center_x, center_y - (n-i)*step)
    svg_finish()
```

# Kompaktnější zápis

```
def star(n = 10, length = 100):  
    svg_header()  
    center_x = length * 1.5  
    center_y = length * 1.5  
    step = length / n  
    for i in range(n + 1):  
        for dx, dy in [(-1,-1), (-1,1), (1,-1), (1,1)]:  
            svg_line(center_x + dx*i*step, center_y,  
                    center_x, center_y + dy*(n-i)*step)  
    svg_finish()
```



# Variace na hvězdu



# Vlastní knihovna pro želví grafiku

- želví grafika – používána knihovna turtle
- vytvořme vlastní “knihovnu” s vykreslováním do SVG
- jen základní příkazy:
  - `forward(length)`
  - `left(angle)`, `right(angle)`
  - `save(filename)`

# Princip implementace

- stav želvy: souřadnice  $x, y$  a aktuální natočení `heading`
- vykreslený obrazec: seznam souřadnic

# Implementace I

```
x = 50
y = 50
heading = 0
lines = []

def left(angle):
    global heading
    heading -= angle

def right(angle):
    global heading
    heading += angle
```

# Implementace II

```
def forward(d):  
    global x  
    global y  
    nx = x + d * math.cos(heading * math.pi / 180)  
    ny = y + d * math.sin(heading * math.pi / 180)  
    lines.append((x, y, nx, ny))  
    x, y = nx, ny
```

jde o názornou ukázkou principů, nikoliv dobrou knihovnu:

- příliš malá funkcionalita
- chybí dokumentace

nevhodné použití globálních proměnných – vhodné pro objektovou reprezentaci

```
class Turtle:
    def __init__(self):
        self.x = 50
        self.y = 50
        self.heading = 0
        self.lines = []

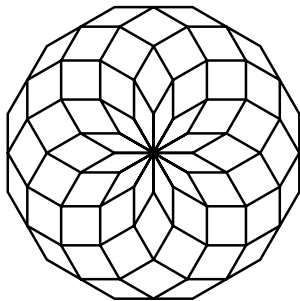
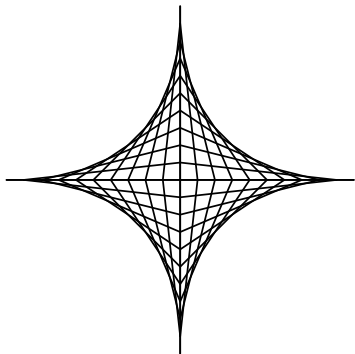
    def left(self, angle):
        self.heading -= angle

    def right(self, angle):
        self.heading += angle

    def forward(self, d):
        nx = self.x + d * math.cos(self.heading * math.pi / 180)
        ny = self.y + d * math.sin(self.heading * math.pi / 180)
        self.lines.append((self.x, self.y, nx, ny))
        self.x, self.y = nx, ny
```

# Absolutní vs relativní vykreslování

(souřadnice vs želva)





- ukázka elementární práce s grafikou
  - bitmapová – Image, putpixel, getpixel
  - vektorová – SVG, line
- využití základních konstrukcí (vesměs vnořené for cykly), trocha matematiky