Data visualization course

**Laboratory work 3**

# Drawing animations in Matplotlib

There are several ways to write Matplotlib code. The following code will be based on the Pyplot interface. Drawing in Matplotlib boils down to adding shapes to the drawing canvas. Listing 1 shows a typical example of a circle image.

*Listing 1 – example*

**import** matplotlib**.**pyplot **as** plt

plt**.**axes**()**

circle **=** plt**.**Circle**((**0**,** 0**),** radius**=**0.75**,** fc**=**'y'**)**

plt**.**gca**().**add\_patch**(**circle**)**

plt**.**axis**(**'scaled'**)**

plt**.**show**()**

The gca() method returns the current Axis instance. Setting the axis to 'scaled' ensures the correct visibility of the added shape. As a result, we will get the image shown in Fig. 1.

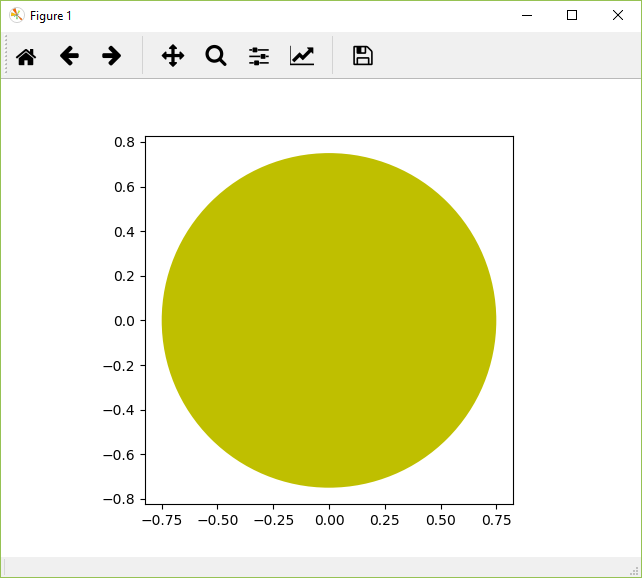


Figure 1 – The result of listing 1

A more detailed drawing example is given in Listing 2. The result of the code is shown in Figure 2.

Listing 2 – Code for drawing a rectangle

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

**import** matplotlib**.**path **as** mpath

**import** matplotlib**.**lines **as** mlines

**import** matplotlib**.**patches **as** mpatches

**from** matplotlib**.**collections **import** PatchCollection

plt**.**rcdefaults**()**

**def** label**(**xy**,** text**):**

y **=** xy**[**1**]** **-** 0.15 # shift y-value for label so that it's below the artist

plt**.**text**(**xy**[**0**],** y**,** text**,** ha**=**"center"**,** family**=**'sans-serif'**,** size**=**14**)**

fig**,** ax **=** plt**.**subplots**()**

# 3x3 grid for drawing objects

grid **=** np**.**mgrid**[**0.2**:**0.8**:**3j**,** 0.2**:**0.8**:**3j**].**reshape**(**2**,** **-**1**).**T

patches **=** **[]**

# circle

circle **=** mpatches**.**Circle**(**grid**[**0**],** 0.1**,** ec**=**"none"**)**

patches**.**append**(**circle**)**

label**(**grid**[**0**],** "Circle"**)**

# rectangle

rect **=** mpatches**.**Rectangle**(**grid**[**1**]** **-** **[**0.025**,** 0.05**],** 0.05**,** 0.1**,** ec**=**"none"**)**

patches**.**append**(**rect**)**

label**(**grid**[**1**],** "Rectangle"**)**

# sector of a circle

wedge **=** mpatches**.**Wedge**(**grid**[**2**],** 0.1**,** 30**,** 270**,** ec**=**"none"**)**

patches**.**append**(**wedge**)**

label**(**grid**[**2**],** "Wedge"**)**

# polygon

polygon **=** mpatches**.**RegularPolygon**(**grid**[**3**],** 5**,** 0.1**)**

patches**.**append**(**polygon**)**

label**(**grid**[**3**],** "Polygon"**)**

# ellipse

ellipse **=** mpatches**.**Ellipse**(**grid**[**4**],** 0.2**,** 0.1**)**

patches**.**append**(**ellipse**)**

label**(**grid**[**4**],** "Ellipse"**)**

# arrow

arrow **=** mpatches**.**Arrow**(**grid**[**5**,** 0**]** **-** 0.05**,** grid**[**5**,** 1**]** **-** 0.05**,** 0.1**,** 0.1**,**

width**=**0.1**)**

patches**.**append**(**arrow**)**

label**(**grid**[**5**],** "Arrow"**)**

# polyline

Path **=** mpath**.**Path

path\_data **=** **[**

**(**Path**.**MOVETO**,** **[**0.018**,** **-**0.11**]),**

**(**Path**.**CURVE4**,** **[-**0.031**,** **-**0.051**]),**

**(**Path**.**CURVE4**,** **[-**0.115**,** 0.073**]),**

**(**Path**.**CURVE4**,** **[-**0.03**,** 0.073**]),**

**(**Path**.**LINETO**,** **[-**0.011**,** 0.039**]),**

**(**Path**.**CURVE4**,** **[**0.043**,** 0.121**]),**

**(**Path**.**CURVE4**,** **[**0.075**,** **-**0.005**]),**

**(**Path**.**CURVE4**,** **[**0.035**,** **-**0.027**]),**

**(**Path**.**CLOSEPOLY**,** **[**0.018**,** **-**0.11**])]**

codes**,** verts **=** zip**(\***path\_data**)**

path **=** mpath**.**Path**(**verts **+** grid**[**6**],** codes**)**

patch **=** mpatches**.**PathPatch**(**path**)**

patches**.**append**(**patch**)**

label**(**grid**[**6**],** "PathPatch"**)**

# a rectangle with rounded edges

fancybox **=** mpatches**.**FancyBboxPatch**(**

grid**[**7**]** **-** **[**0.025**,** 0.05**],** 0.05**,** 0.1**,**

boxstyle**=**mpatches**.**BoxStyle**(**"Round"**,** pad**=**0.02**))**

patches**.**append**(**fancybox**)**

label**(**grid**[**7**],** "FancyBboxPatch"**)**

# Line

x**,** y **=** np**.**array**([[-**0.06**,** 0.0**,** 0.1**],** **[**0.05**,** **-**0.05**,** 0.05**]])**

line **=** mlines**.**Line2D**(**x **+** grid**[**8**,** 0**],** y **+** grid**[**8**,** 1**],** lw**=**5.**,** alpha**=**0.3**)**

label**(**grid**[**8**],** "Line2D"**)**

colors **=** np**.**linspace**(**0**,** 1**,** len**(**patches**))**

collection **=** PatchCollection**(**patches**,** cmap**=**plt**.**cm**.**hsv**,** alpha**=**0.3**)**

collection**.**set\_array**(**np**.**array**(**colors**))**

ax**.**add\_collection**(**collection**)**

ax**.**add\_line**(**line**)**

plt**.**axis**(**'equal'**)**

plt**.**axis**(**'off'**)**

plt**.**tight\_layout**()**

plt**.**show**()**

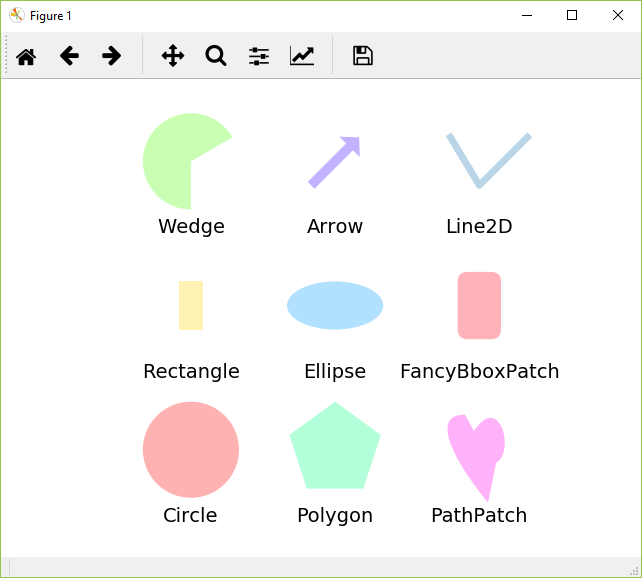


Figure 2 – The result of listing 2

# Animation

When creating animation, the greatest interest lies in the movement of some figures along a given trajectory. Listing 3 shows the code that animates circle by circle:

Listing 3 – Animation of figure movement

**import** numpy **as** np

**from** matplotlib **import** pyplot **as** plt

**from** matplotlib **import** animation

fig **=** plt**.**figure**()**

fig**.**set\_dpi**(**100**)**

fig**.**set\_size\_inches**(**7**,** 6.5**)**

ax **=** plt**.**axes**(**xlim**=(**0**,** 10**),** ylim**=(**0**,** 10**))**

patch **=** plt**.**Circle**((**5**,** **-**5**),** 0.75**,** fc**=**'y'**)**

**def** init**():**

patch**.**center **=** **(**5**,** 5**)**

ax**.**add\_patch**(**patch**)**

**return** patch**,**

**def** animate**(**i**):**

x**,** y **=** patch**.**center

x **=** 5 **+** 3 **\*** np**.**sin**(**np**.**radians**(**i**))**

y **=** 5 **+** 3 **\*** np**.**cos**(**np**.**radians**(**i**))**

patch**.**center **=** **(**x**,** y**)**

**return** patch**,**

anim **=** animation**.**FuncAnimation**(**fig**,** animate**,**

init\_func**=**init**,**

frames**=**360**,**

interval**=**20**,**

blit**=True)**

plt**.**show**()**

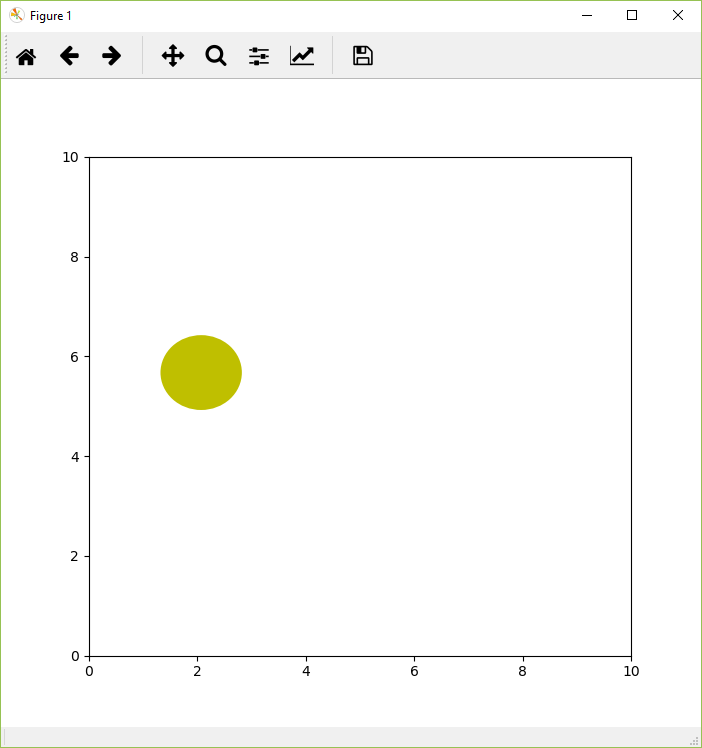
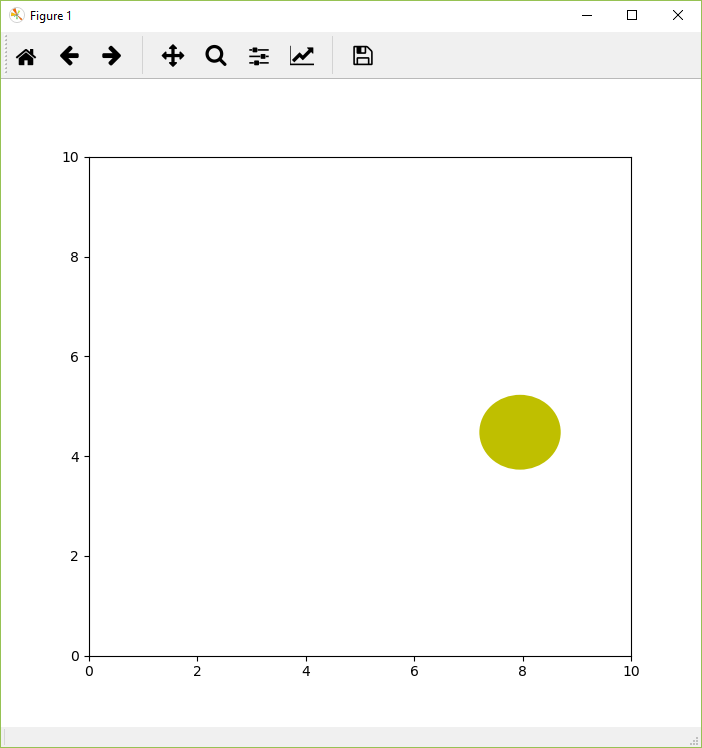


Figure 3 – Results of animation construction

To save the animation to disk, you can use the following code:

Listing 4 – Saving the animation

anim**.**save**(**'animation.mp4'**,** fps**=**30**,**

extra\_args**=[**'-vcodec'**,** 'h264'**,**

'-pix\_fmt'**,** 'yuv420p'**])**

Unfortunately, matplotlib does not have the ability to save in gif format, so you can use ImageMagic to convert a set of pictures into a gif animation:

Listing 5 – using ImageMagic to create animations

ffmpeg -i animation.mp4 -r 10 output%05d.png

convert output\*.png output.gif

# Task

Develop a program that builds animation according to the option, save it in gif format. The options are set according to the student number:

1. A sector of a circle moving along a rectangle and changing the angle of the sector;
2. An arrow, one end of which moves along the rectangle, and the other constantly points to the center of the window;
3. Polyline in the form of a tick (V) moving along an equilateral triangle;
4. An ellipse rotating around its own center;
5. A rectangle whose center moves along an elliptical trajectory;
6. A rectangle with rounded edges moving along a sine wave;
7. A circle moving in a spiral;
8. A polygon in the shape of a pentagon moving in a circle;
9. Expanding/Decreasing heart