Programowanie w Pythonie

Łukasz Mioduszewski, UKSW 2022

Biblioteka opencv, moduł matplotlib.animation

matplotlib.animation

- Wszystko co pojawia się na obrazku (klasa Figure) jest obiektem dziedziczącym z Artist – niektóre podklasy tej klasy są animowane
- Jeśli korzystamy z takich animowanych obiektów, używamy ArtistAnimation
- Jeśli chcemy w każdej klatce rysować od nowa, używamy FuncAnimation

matplotlib.animation

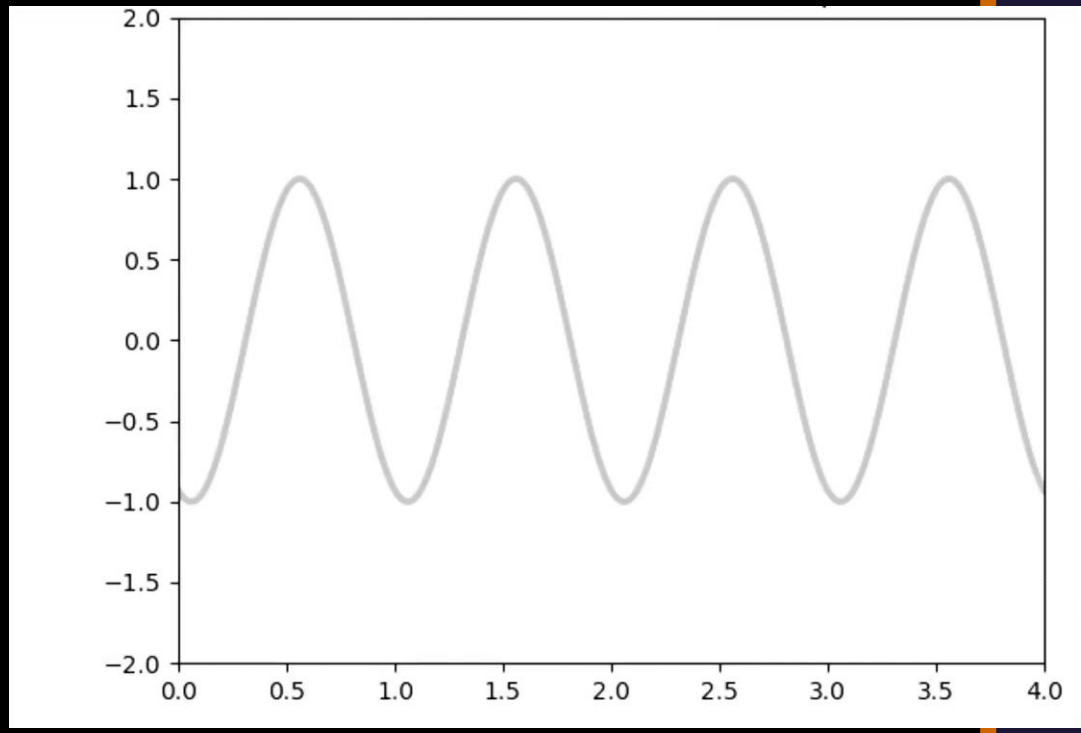
- Zawsze trzeba mieć referencję do animowanego obiektu, inaczej zniknie (garbage collection)
- FuncAnimation jest prostsze, bo sami wybieramy co rysujemy
  - Wada: trzeba za każdym razem rysować wszystko od nowa… choć niekoniecznie
- Blitting to technika która rysuje od nowa tylko te obiekty które zmieniły się pomiędzy klatkami (parametr FuncAnimation blit=True)
- Aby zapisywać animacje do pliku, potrzeba ffmpeg, ale nie wystarczy zwykłe: pip install ffmpeg-python https://pypi.org/project/ffmpeg-python/ potrzeba samego ffmpeg w PATH https://www.ffmpeg.org/download.html

animatetutorial.py

```
from matplotlib import pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation

# initializing a figure in which the graph will be plotted
fig = plt.figure()
# marking the x-axis and y-axis
axis = plt.axes(xlim =(0, 4), ylim =(-2, 2))
# initializing a line variable
line, = axis.plot([], [], lw = 3)

# data which the line will contain (x, y)
def init():
    line.set_data([], [])
    return line,

def animate(i):
    x = np.linspace(0, 4, 1000)
    # plots a sine graph
    y = np.sin(2 * np.pi * (x - 0.01 * i))
    line.set_data(x, y)
    return line,

anim = FuncAnimation(fig, animate, init_func = init,
                     frames = 200, interval = 20, blit = True)
anim.save('continuousSineWave.mp4', writer = 'ffmpeg', fps = 30)
```

animatetutorial.py

```
1  from matplotlib import pyplot as plt
2  import numpy as np
3  from matplotlib.animation import FuncAnimation
4
5  # initializing a figure in which the graph will be plotted
6  fig = plt.figure()
7  # marking the x-axis and y-axis
8  axis = plt.axes(xlim =(0, 4), ylim =(-2, 2))
9  # initializing a line variable
10 line, = axis.plot([], [], lw = 3)
11
12 # data which the line will contain (x, y)
13 def init():
14     line.set_data([], [])
15     return line,
16
17 def animate(i):
18     x = np.linspace(0, 4, 1000)
19     # plots a sine graph
20     y = np.sin(2 * np.pi * (x - 0.01 * i))
21     line.set_data(x, y)
22     return line,
23
24 anim = FuncAnimation(fig, animate, init_func = init,
25                      frames = 200, interval = 20, blit = True)
26 anim.save('continuousSineWave.mp4', writer = 'ffmpeg', fps = 30)
```



animatetutorial2.py

```
1  import matplotlib.animation as animation
2  import matplotlib.pyplot as plt
3  import numpy as np
4  # creating a blank window for the animation
5  fig = plt.figure()
6  axis = plt.axes(xlim =(-50, 50), ylim =(-50, 50))
7  line, = axis.plot([], [], lw = 2)
8  def init():
9     line.set_data([], [])
10    return line,
11 # initializing empty values for x and y co-ordinates
12 xdata, ydata = [], []
13
14 def animate(i):
15    # t is a parameter which varies with the frame number
16    t = 0.1 * i
17    x, y = t * np.sin(t), t * np.cos(t)
18    # appending values to the previously empty x and y data holders
19    xdata.append(x)
20    ydata.append(y)
21    line.set_data(xdata, ydata)
22    return line,
23 # calling the animation function
24 anim = animation.FuncAnimation(fig, animate, init_func = init,
25                                frames = 500, interval = 20, blit = True)
26 plt.show()
```

animatetutorial2.py

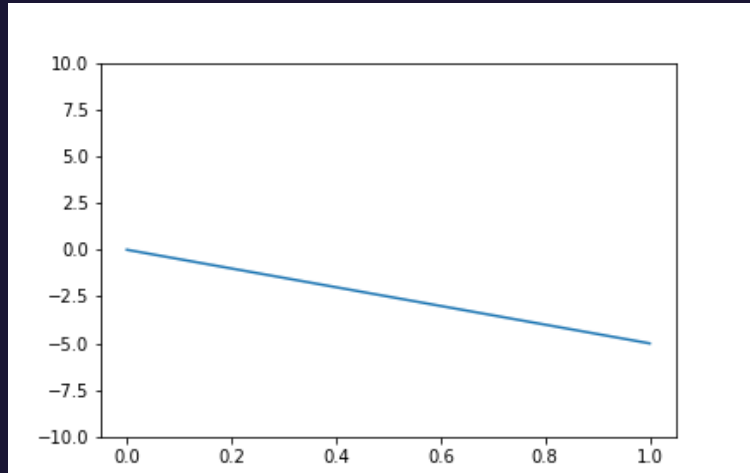
```
1  import matplotlib.animation as animation
2  import matplotlib.pyplot as plt
3  import numpy as np
4  # creating a blank window for the animation
5  fig = plt.figure()
6  axis = plt.axes(xlim =(-50, 50), ylim =(-50, 50))
7  line, = axis.plot([], [], lw = 2)
8  def init():
9     line.set_data([], [])
10    return line,
11 # initializing empty values for x and y
12 xdata, ydata = [], []
13
14 def animate(i):
15    # t is a parameter which varies with
16    t = 0.1 * i
17    x, y = t * np.sin(t), t * np.cos(t)
18    # appending values to the previously
19    xdata.append(x)
20    ydata.append(y)
21    line.set_data(xdata, ydata)
22    return line,
23 # calling the animation function
24 anim = animation.FuncAnimation(fig, animate, init_func = init,
25                                frames = 500, interval = 20, blit = True)
26 plt.show()
```

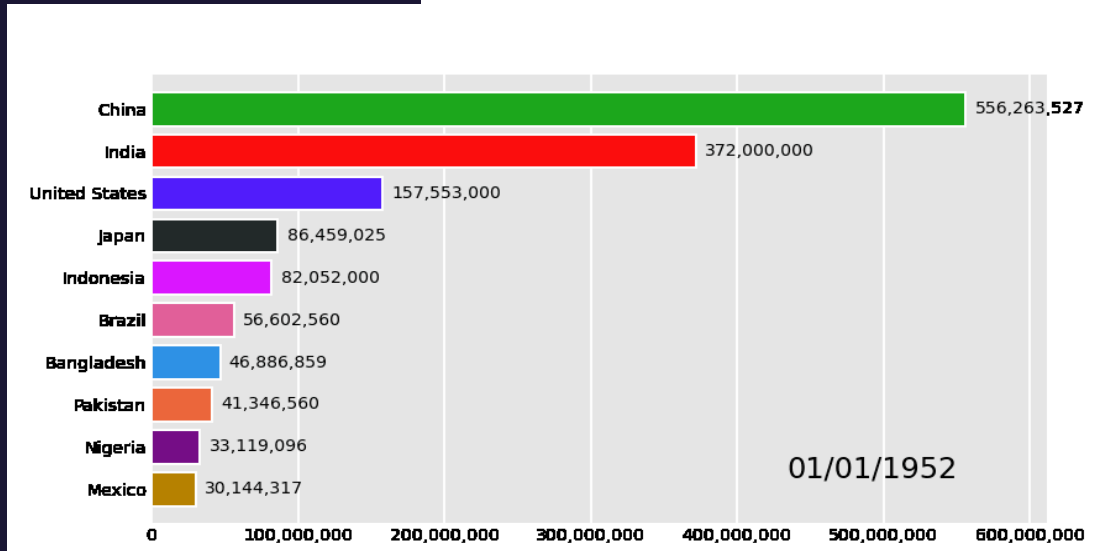
animatetutorial3.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = ax.plot([], [], 'ro')

def init():
    ax.set_xlim(0, 2*np.pi)
    ax.set_ylim(-1, 1)
    return ln,

def update(frame):
    xdata.append(frame)
    ydata.append(np.sin(frame))
    ln.set_data(xdata, ydata)
    return ln,

ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*np.pi, 128),
                    init_func=init, blit=True)
plt.show()
```

matplotlib.animation - alternatywy

- ImageIO
  https://towardsdatascience.com/probably-the-easiest-way-to-animate-your-python-plots-f5194ebed75f
- pandas_alive
  https://betterprogramming.pub/this-python-library-can-animate-your-charts-a7c0a98b3463
- celluloid (zapis do gif)





Wstęp do openCV - plan

- Instalacja: pip install opencv-python
- Wczytanie obrazka
- Odczytanie wartości RGB pikseli
- Wycinanie interesującego nas fragmentu
- Zmiana rozmiaru i obrót obrazka
- Rysowanie prostokątów
- Wyświetlanie tekstu

Wczytywanie obrazka

openCVtutorial.py

```
1  # Importing the OpenCV library
2  import cv2
3  # Reading the image using imread() function
4  image = cv2.imread('image.png')
5
6  # Extracting the height and width of an image
7  h, w = image.shape[:2]
8  # Displaying the height and width
9  print("Height = {},  Width = {}".format(h, w))
```

Wartości RGB pikseli

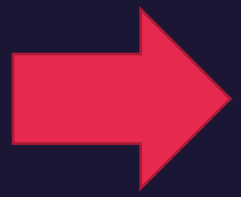
openCVtutorial.py

```
10
11 # pixel with 100, 100 for height and width.
12 (B, G, R) = image[100, 100]
13
14 # Displaying the pixel values
15 print("R = {}, G = {}, B = {}".format(R, G, B))
16
17 # We can also pass the channel to extract
18 # the value for a specific channel
19 B = image[100, 100, 0]
20 print("B = {}".format(B))
```

Wycinanie fragmentu obrazka

openCVtutorial.py

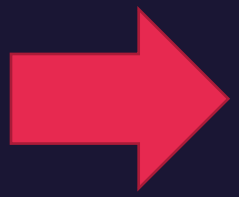
```
21
22  # slicing the pixels of the image
23  roi = image[100 : 500, 200 : 700]
```

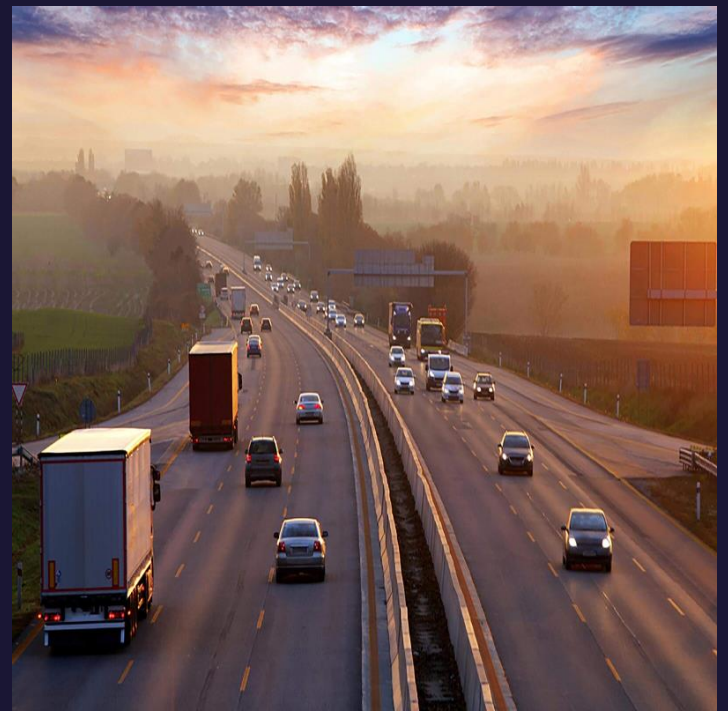


Zmiana rozmiaru obrazka

openCVtutorial.py

```
24  # resize() function takes 2 parameters,
25  # the image and the dimensions
26  resize = cv2.resize(image, (800, 800))
```

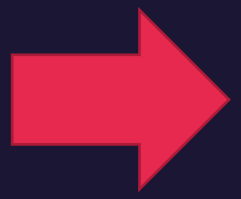




Zmiana rozmiaru obrazka

openCVtutorial.py

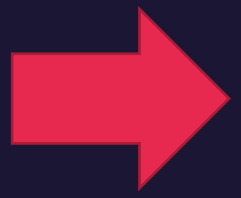
```
27
28  # Calculating the ratio
29  ratio = 800 / w
30
31  # Creating a tuple containing width and height
32  dim = (800, int(h * ratio))
33  # Resizing the image
34  resize_aspect = cv2.resize(image, dim)
```



Obrót obrazka

openCVtutorial.py

```
35
36  # Calculating the center of the image
37  center = (w // 2, h // 2)
38  # Generating a rotation matrix
39  matrix = cv2.getRotationMatrix2D(center, -45, 1.0)
40
41  # Performing the affine transformation
42  rotated = cv2.warpAffine(image, matrix, (w, h))
```



Obrót obrazka

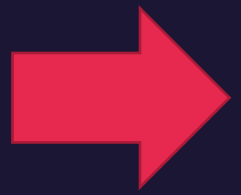
- getRotationMatrix2D() potrzebuje 3 argumentów:
  - center – Współrzędne osi obrotu (zwykle środek obrazka)
  - Angle – kąt obrotu (w stopniach)
  - Scale – czynnik skalujący
- Zwraca macierz 2x3, której wartości zależą od:
- alpha = scale * cos(angle)
- beta = scale * sine(angle)

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \texttt{center.x} - \beta \cdot \texttt{center.y} \\ -\beta & \alpha & \beta \cdot \texttt{center.x} + (1-\alpha)\cdot \texttt{center.y} \end{bmatrix}$$

Rysowanie prostokąta

openCVtutorial.py

```
43
44  # We are copying the original image,
45  # as it is an in-place operation.
46  output = image.copy()
47  # Using the rectangle() function to create a rectangle.
48  rectangle = cv2.rectangle(output, (1500, 900),
49                            (600, 400), (255, 0, 0), 2)
```



Rysowanie prostokąta

- rectangle() potrzebuje 5 argumentów:
  - obrazka
  - współrzędnych górnego lewego rogu prostokąta
  - współrzędnych dolnego prawego rogu prostokąta
  - koloru (w formacie BGR)
  - grubości linii

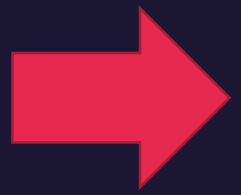
openCVtutorial.py

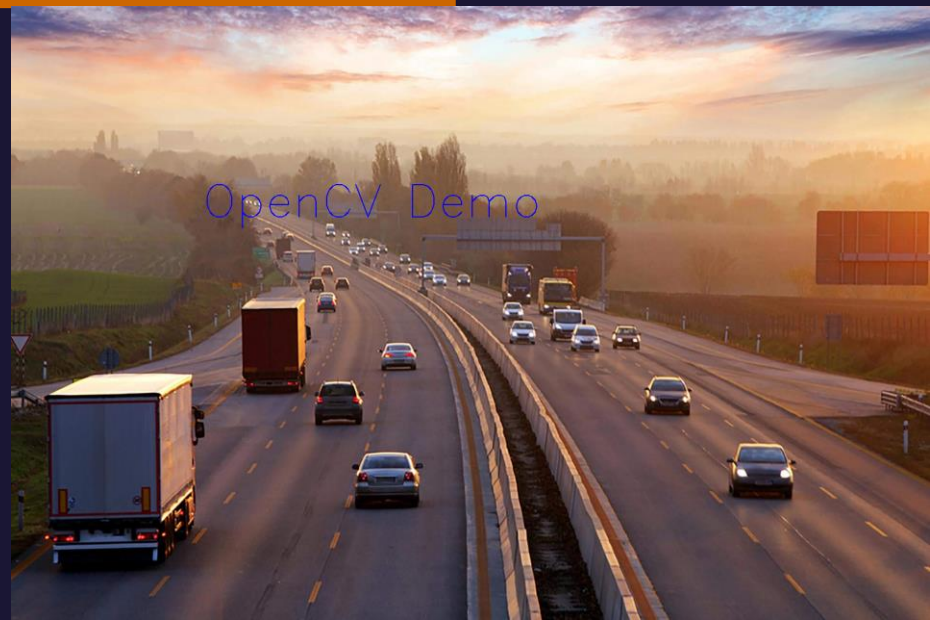
```
43
44  # We are copying the original image,
45  # as it is an in-place operation.
46  output = image.copy()
47  # Using the rectangle() function to create a rectangle.
48  rectangle = cv2.rectangle(output, (1500, 900),
49                            (600, 400), (255, 0, 0), 2)
```

Dodawanie tekstu

openCVtutorial.py

```
50  # Copying the original image
51  output = image.copy()
52
53  # Adding the text using putText() function
54  text = cv2.putText(output, 'OpenCV Demo', (500, 550),
55                     cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
```





Dodawanie tekstu

- putText() potrzebuje 7 argumentów:
  - obrazka
  - tekstu
  - współrzędnych dolnego lewego rogu prostokąta
  - kroju pisma
  - rozmiaru liter
  - koloru
  - grubości linii

openCVtutorial.py

```
50  # Copying the original image
51  output = image.copy()
52
53  # Adding the text using putText() function
54  text = cv2.putText(output, 'OpenCV Demo', (500, 550),
55                     cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
```

Bardziej skomplikowane przykłady...

- Animacja planet krążących wokół słońca i księżyca wokół ziemi
- Skrypt do pokazu naukowego ilustrującego ideę FCS (Fluorescent Correlation Spectroscopy) poprzez filmowanie czerwonych i niebieskich kulek w pojemniku ze wzburzoną wodą
- Dostępne na Moodle

Bibliografia

- https://www.geeksforgeeks.org/using-matplotlib-for-animations/
- https://www.geeksforgeeks.org/introduction-to-opencv/