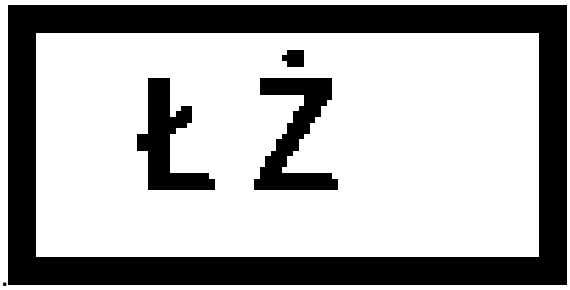
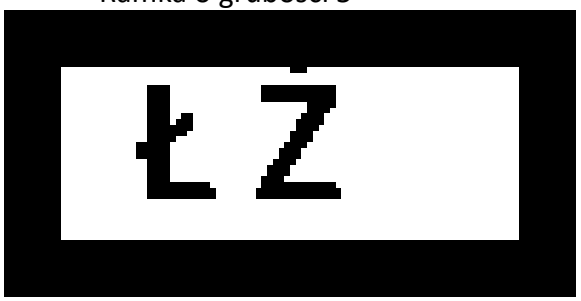
Obrazy w trybie 1

```python
import numpy as np
from PIL import Image

def rysuj_ramki_w_obrazie(tab_obrazu, grubosc, odstep = 0):
    h, w = tab_obrazu.shape
    start = 0
    while start < min(h, w) / 2:
        for i in range(start, h - start):
            if start + grubosc < h:
                for j in range(start, grubosc + start):
                    tab_obrazu[i][j] = 0
                for j in range(w - grubosc - start, w - start):
                    tab_obrazu[i][j] = 0
        for i in range(start, w - start):
            if start + grubosc < w:
                for j in range(start, grubosc + start):
                    tab_obrazu[j][i] = 0
                for j in range(h - grubosc - start, h - start):
                    tab_obrazu[j][i] = 0
        if odstep == 0:
            tab = tab_obrazu.astype(bool)
            return Image.fromarray(tab)
        else:
            start += odstep + grubosc
    tab = tab_obrazu.astype(bool)
    return Image.fromarray(tab)

image = Image.open("inicjaly.bmp")
tablica_obrazu = np.asarray(image, dtype=bool) * 1

ramki5 = rysuj_ramki_w_obrazie(tablica_obrazu, grubosc=5)
ramki10 = rysuj_ramki_w_obrazie(tablica_obrazu, grubosc=10)

ramki5.save("ramka5.bmp")
ramki10.save("ramka10.bmp")
```

Wczytałem obraz inicjały 0 rozmiarach 150x50, następnie po zamianie go na tablicę podałem jako argument do funkcji. Funkcja ta działa dwojako, nie podając parametru odstęp automatycznie przypisuje do nie wartość 0. W taki sposób funkcja zrobi po prostu ramkę w obrazie. Na tym przykładzie rysowana jest ramka o grubości 5 i 10 widoczne poniżej



Ramka o grubości 5

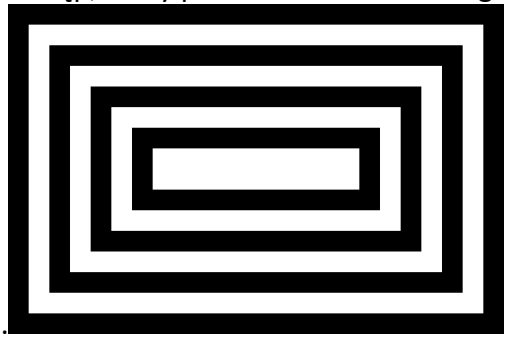


Ramka o grubości 10

Ten sam kod spełnia funkcje tworzenia ramek na przemian, że raz jest biała raz czarna. Jedynie różni się, że zamiast wczytywać i przekształcać obrazek, od razu tworzę tablice:

```python
def rysuj_ramki_w_obrazie(h, w, grubosc, odstep = 0):
    tab_obrazu = np.ones((h, w))
```

W tym przypadku obraz jest cały biały, a czarne ramki są po prostu dodawane poprzez zmianę punktu startowego. Parametry jakie przyjąłem są takie same jak w poprzednim przykładzie, ale tu zastosowałem dodatkowy parametr odstęp, który pozwala sterować odległością pomiędzy ramkami. Parametry: h=320. W=480, grubość=20, odstep=20



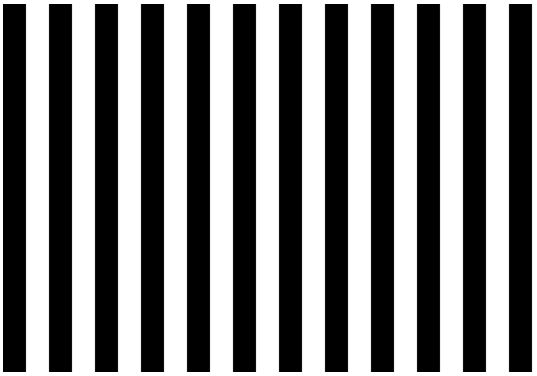
Ramki o grubości 20 i odstępie 20

```python
import numpy as np
from PIL import Image


def rysuj_ramki_w_obrazie(h, w, grubosc, odstep = 0):
    tab_obrazu = np.ones((h, w))
    start = 0
    while start < min(h, w) / 2:
        for i in range(start, h - start):
            if start + grubosc < h:
                for j in range(start, grubosc + start):
                    tab_obrazu[i][j] = 0
                for j in range(w - grubosc - start, w - start):
                    tab_obrazu[i][j] = 0
        for i in range(start, w - start):
            if start + grubosc < w:
                for j in range(start, grubosc + start):
                    tab_obrazu[j][i] = 0
                for j in range(h - grubosc - start, h - start):
                    tab_obrazu[j][i] = 0
        if odstep == 0:
            tab = tab_obrazu.astype(bool)
            return Image.fromarray(tab)
        else:
            start += odstep + grubosc
    tab = tab_obrazu.astype(bool)
    return Image.fromarray(tab)

ramki = rysuj_ramki_w_obrazie(h=320, w=480, grubosc=20, odstep=20)
ramki.save("ramki_w_srodku.bmp")
```

Jak widać, obie funkcje niczym się nie różnią, poza podanymi argumentami. W razie potrzeby można zmodyfikować kod, i dać możliwość zmiany takich parametrów użytkownikowi.

```python
import numpy as np
from PIL import Image


def rysuj_paski_w_obrazie(h, w, grubosc, odstep = 10):
    tab_obraz = np.ones((h, w))
    start = 0
    while start < max(h, w):
        for i in range(0, h):
            if start + grubosc < w:
                for j in range(start, grubosc + start):
                    tab_obraz[i][j] = 0
        start += odstep + grubosc
    tab = tab_obraz.astype(bool)
    return Image.fromarray(tab)


paski = rysuj_paski_w_obrazie(h=320, w=480, grubosc=20, odstep=20)
paski.save("paski.png")
```

Funkcja ta rysuje czarne pasy na białym tle. Jest możliwość ustawienia odstępu między tymi pasami, jednak domyślnie jest ustawiony na 10, gdyż przy wielkości 0 wyszedłby po prostu czarny obraz. Przyjęte parametry to:
h=320, w=480, grubość=20, odstęp=20

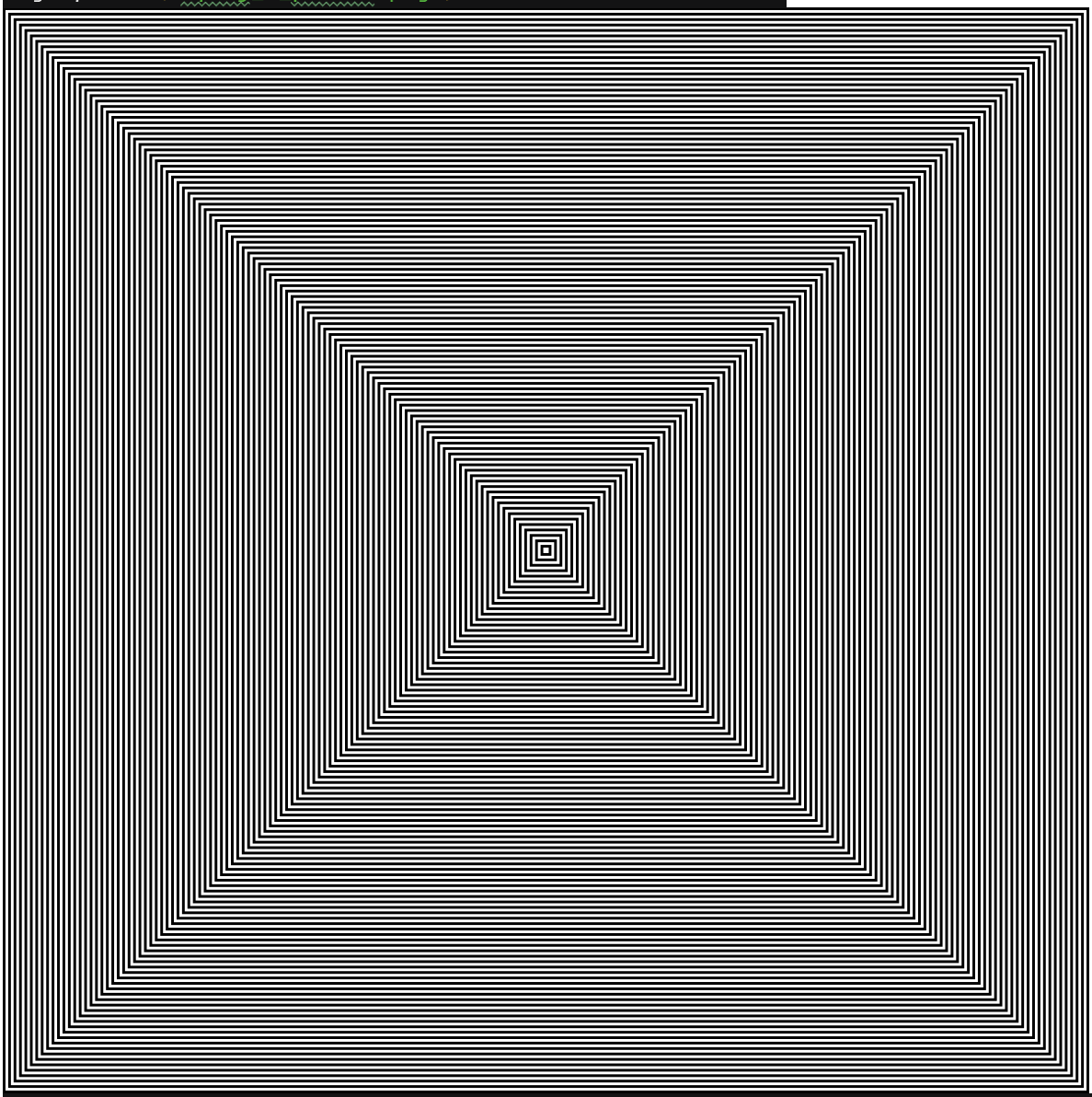


Paski o grubości 20 i odstępie 20

```python
import numpy as np
from PIL import Image

def kwadraty(h, w, h_start, w_start):
    tab_obrazu = np.ones((h, w))
    h, w = tab_obrazu.shape
    for i in range(h_start, -1, -1):
        for j in range(w_start, -1, -1):
            tab_obrazu[i][j]=0
    for i in range(h_start, h):
        for j in range(w_start, w):
            tab_obrazu[i][j]=0
    tab = tab_obrazu.astype(bool)
    return Image.fromarray(tab)

figury = kwadraty(h=320, w=480, h_start=50, w_start= 100)
figury.save("rysuj_z_punktu.png")
```

Ta funkcja rysuje dwa czarne prostokąty stykające się w podanym punkcie. Jako parametry przyjęto h=320, w=480 i współrzędne startowe nazwane tu h_start=50 oraz w_start=100.

Prostokąty stykające się w punkcie 100x50



```python
ramki = rysuj_ramki_w_obrazie(h=2000, w=2000, grubosc=5, odstep=5)
ramki.save("wlasny.bmp")
```

Obraz o parametrach powyżej, ciekawe złudzenie, jakby były tam różne kolory, a cały obraz to po prostu coraz mniejsze ramki.

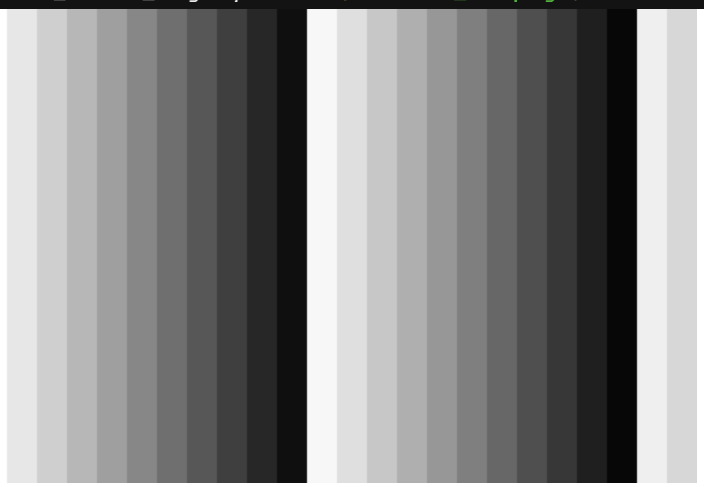
Obrazy w trybie L

```python
import numpy as np
from PIL import Image


def rysuj_szare_paski(h, w, grubosc):
    tab_obrazu = np.zeros((h, w), dtype=np.uint8)
    start = 0
    odcien = np.uint8(255)
    while start < max(h, w):
        for i in range(0, h):
            if start + grubosc <= w:
                for j in range(start, grubosc + start):
                    tab_obrazu[i][j] = odcien
        start += grubosc
        if odcien < 0:
            odcien = np.uint8(255)
        else:
            odcien = np.uint8(odcien - max(h, w) // grubosc)
    tab = tab_obrazu.astype(np.uint8)
    return Image.fromarray(tab)

ramki_szare = rysuj_szare_paski(h=320, w=480, grubosc= 20)
ramki_szare_negatyw = Image.fromarray(255 - np.asarray(ramki_szare))
ramki_szare.save("obraz1_1.png")
ramki_szare_negatyw.save("obraz1_1N.png")
```

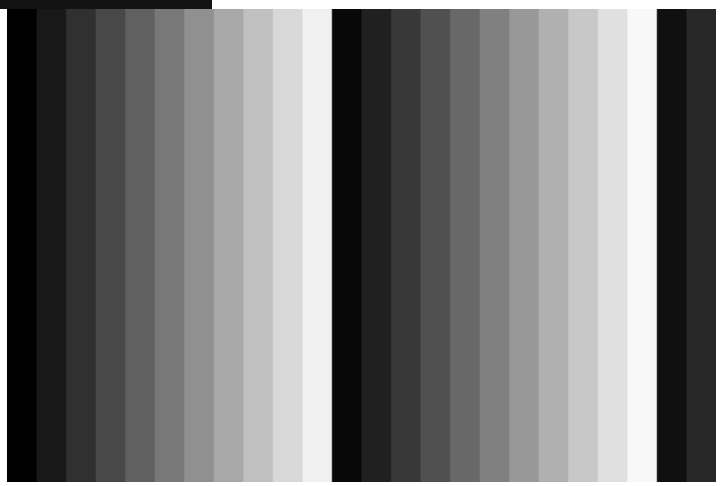
Funkcja tworzy pasy w odcieniach szarości o wybranej grubości. Na początku przyjęto, że pierwszy pas będzie biały, a kiedy wartość odcienia spadnie poniżej zera zostanie ponownie ustawiony na wartość 255. W innych warunkach wartość odcienia wyliczana będzie według wzoru:

```python
odcien = np.uint8(odcien - max(h, w) // grubosc)
```

Czyli przy każdym przejściu pętli obecna wartość odcienia będzie pomniejszana o wynik dzielenia bez reszty maksymalnej wartości h lub w podzielonej przez ustaloną grubość.



Obraz o parametrach h=320, w=480, grubość=20



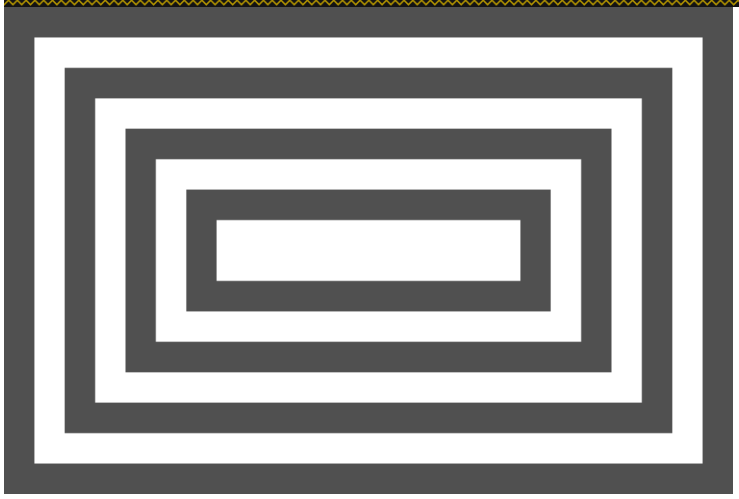
Negatyw obrazu obok

```python
import numpy as np
from PIL import Image


def rysuj_szare_ramki_w_obrazie(h, w, grubosc, odstep = 0, odcien = 0):
    tab_obrazu = np.full((h, w), 255, dtype=np.uint8)
    start = 0
    odcien = np.uint8(odcien)
    while start < min(h, w) / 2:
        for i in range(start, h - start):
            if start + grubosc < h:
                for j in range(start, grubosc + start):
                    tab_obrazu[i][j] = odcien
                for j in range(w - grubosc - start, w - start):
                    tab_obrazu[i][j] = odcien
        for i in range(start, w - start):
            if start + grubosc < w:
                for j in range(start, grubosc + start):
                    tab_obrazu[j][i] = odcien
                for j in range(h - grubosc - start, h - start):
                    tab_obrazu[j][i] = odcien
        if odstep == 0:
            tab = tab_obrazu.astype(np.uint8)
            return Image.fromarray(tab)
        else:
            start += odstep + grubosc
        tab = tab_obrazu.astype(np.uint8)
    tab = tab_obrazu.astype(np.uint8)
    return Image.fromarray(tab)

ramki_szare = rysuj_szare_ramki_w_obrazie(h=320, w=480, grubosc=20, odstep=20, odcien=80)
ramki_szare_negatyw = Image.fromarray(255 - np.asarray(ramki_szare))
ramki_szare.save("obraz1_2.png")
ramki_szare_negatyw.save("obraz1_2N.png")
```

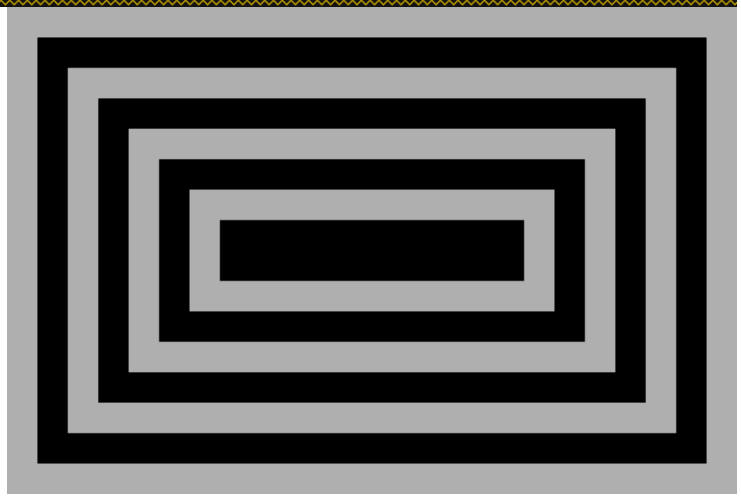
Kod rysuje w zależności od parametru odstęp albo ramkę, albo wiele ramek coraz mniejszych oddalonych od siebie o wybrany parametr odstep i o wybranej grubości.

Przykładowy wynik i jego negatyw przy podanych parametrach. Tym razem odcień ustawiono jako stałą wartość i nie zmienia się ona w programie.

```python
ramki_szare = rysuj_szare_ramki_w_obrazie(h=320, w=480, grubosc=20, odstep=20, odcien=80)
```



Obraz powstały przy podanych parametrach

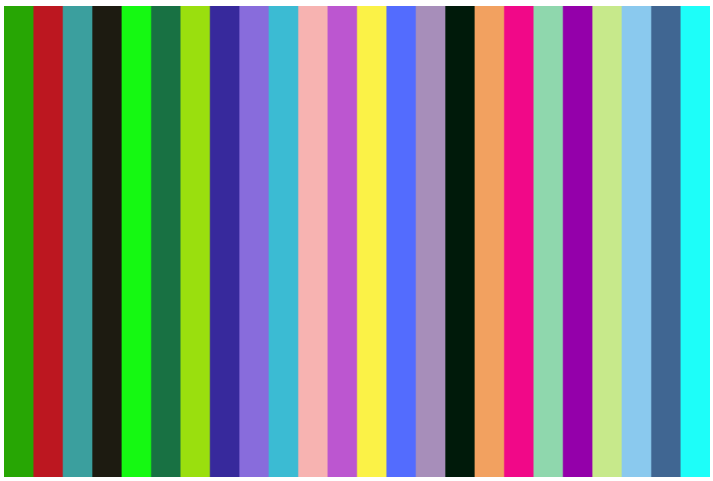


Negatyw tego obrazu

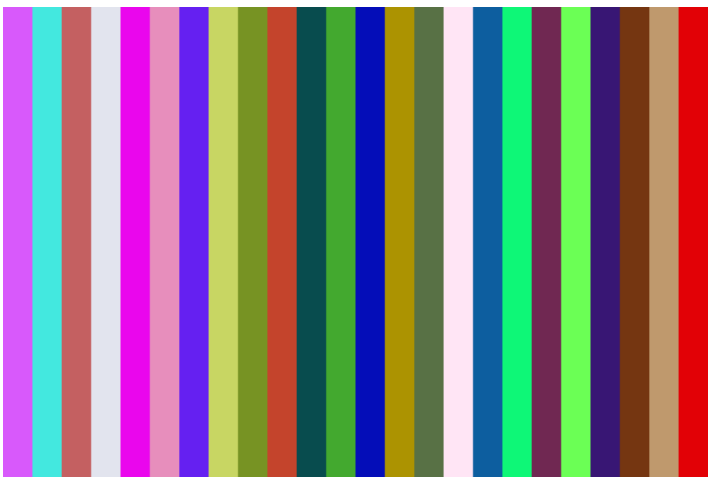
Obrazy w trybie RGB

```
import numpy as np
from PIL import Image
from random import randint


def rysuj_paski_rgb(h, w, grubosc):
    tab_rgb = np.full((h, w, 3), [0, 0, 0], dtype=np.uint8)
    start = 0

    while start < max(h, w):
        kolor_r = randint(0, 256)
        kolor_g = randint(0, 256)
        kolor_b = randint(0, 256)
        for i in range(0, h):
            if start + grubosc <= w:
                for j in range(start, grubosc + start):
                    tab_rgb[i][j] = [kolor_r, kolor_g, kolor_b]
        start += grubosc
    tab = tab_rgb.astype(np.uint8)
    return Image.fromarray(tab)


paski_rgb = rysuj_paski_rgb(h=320, w=480, grubosc= 20)
paski_rgb_negatyw = Image.fromarray(255 - np.asarray(paski_rgb))
paski_rgb.save("obraz2_1.png")
paski_rgb.save("obraz2_1.jpg")
paski_rgb_negatyw.save("obraz2_1N.png")
paski_rgb_negatyw.save("obraz2_1N.jpg")
```

Funkcja podobnie jak rysująca szare pasy rysuje tym razem kolorowe RGB o podanej przez użytkownika grubości. Aby wybrać zaimportowano funkcję randint z biblioteki random i ustalono aby wybierał losowo z zakresu 0-256. To sprawi, że po narysowaniu danego pasa kolor zmienia się sposób losowy, a każde wywołanie programu daje różne wyniki. Przyjęto następujące parametry: h=320, w=480, grubość=20



Obraz powstały z podanych parametrów.

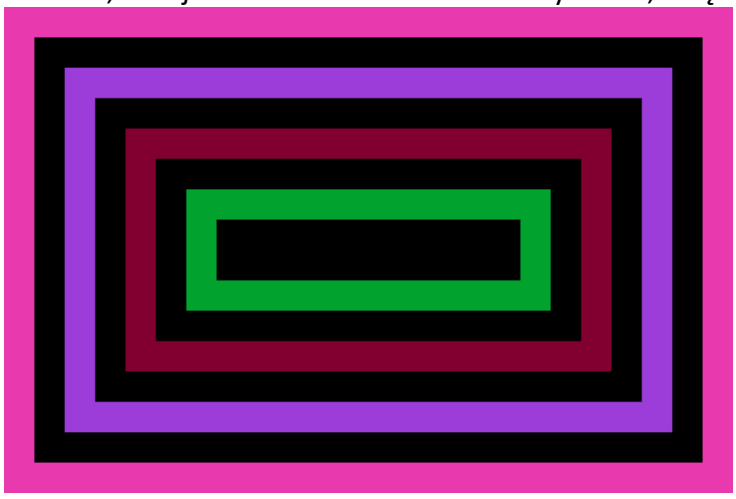


Negatyw tego obrazu

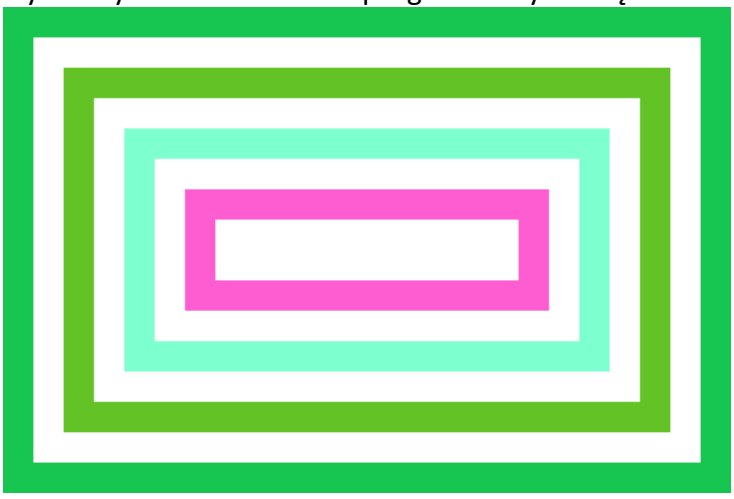
..

```python
import numpy as np
from PIL import Image
from random import randint


def rysuj_ramki_rgb_w_obrazie(h, w, grubosc, odstep = 0):
    tab_rgb = np.full((h, w, 3), [0, 0, 0], dtype=np.uint8)
    start = 0
    while start < min(h, w) / 2:
        kolor_r = randint(0, 256)
        kolor_g = randint(0, 256)
        kolor_b = randint(0, 256)
        for i in range(start, h - start):
            if start + grubosc < h:
                for j in range(start, grubosc + start):
                    tab_rgb[i][j] = [kolor_r, kolor_g, kolor_b]
                for j in range(w - grubosc - start, w - start):
                    tab_rgb[i][j] = [kolor_r, kolor_g, kolor_b]
        for i in range(start, w - start):
            if start + grubosc < w:
                for j in range(start, grubosc + start):
                    tab_rgb[j][i] = [kolor_r, kolor_g, kolor_b]
                for j in range(h - grubosc - start, h - start):
                    tab_rgb[j][i] = [kolor_r, kolor_g, kolor_b]
        if odstep == 0:
            tab = tab_rgb.astype(np.uint8)
            return Image.fromarray(tab)
        else:
            start += odstep + grubosc
        tab = tab_rgb.astype(np.uint8)
    tab = tab_rgb.astype(np.uint8)
    return Image.fromarray(tab)

ramki_rgb = rysuj_ramki_rgb_w_obrazie(h=320, w=480, grubosc=20, odstep=20)
ramki_rgb_negatyw = Image.fromarray(255 - np.asarray(ramki_rgb))
ramki_rgb.save("obraz2_2.png")
ramki_rgb.save("obraz2_2.jpg")
ramki_rgb_negatyw.save("obraz2_2N.png")
ramki_rgb_negatyw.save("obraz2_2N.jpg")
```

Program rysuje ramkę przy parametrze odstep = 0 lub wiele ramek kiedy odstęp jest > 0. Podobnie jak poprzednio do ustalenia koloru zastosowano funkcję randint z bibliotemi random o parametrach 0-256. W odróżnieniu od odcieni szarości, tutaj każda ramka ma losowany kolor, więc przy każdym uruchomieniu programu wynik będzie inny.



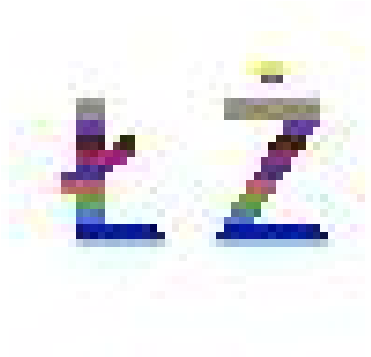
Obraz powstały przy podanych parametrach

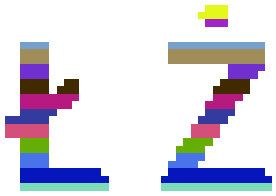


Negatyw tego obrazu

```python
import numpy as np
from PIL import Image
from random import randint

def konwertuj_rgb(tab_obrazu):
    for i in range(tab_obrazu.shape[0]):
        for j in range(tab_obrazu.shape[1]):
            if np.array_equal(tab_obrazu[i][j], [1, 1, 1]):
                tab_obrazu[i][j] = [255, 255, 255]
            else:
                tab_obrazu[i][j] = [0, 0, 0]
    return tab_obrazu

def rysuj_paski_rgb(inicjaly_rgb, grubosc):
    start = 0
    while start < min(inicjaly_rgb.shape[0], inicjaly_rgb.shape[1]):
        kolor_r = randint(0, 256)
        kolor_g = randint(0, 256)
        kolor_b = randint(0, 256)
        for i in range(0, inicjaly_rgb.shape[1]):
            if start + grubosc ⩽ inicjaly_rgb.shape[0]:
                for j in range(start, grubosc + start):
                    if np.array_equal(inicjaly_rgb[j][i], [0, 0, 0]):
                        inicjaly_rgb[j][i] = [kolor_r, kolor_g, kolor_b]
        start += grubosc
    return inicjaly_rgb


image = Image.open("inicjaly.bmp")

image_array = np.array(image, dtype=np.uint8)
image_rgb = np.stack((image_array, image_array, image_array), axis=-1)

obraz_rgb = Image.fromarray(rysuj_paski_rgb(konwertuj_rgb(image_rgb), 2))

obraz_rgb.save("inicjaly_jpg.jpg")
obraz_rgb.save("inicjaly_png.png")
```

W powyższym kodzie wczytujemy w pierwszej kolejności znany z poprzednich lab1 plik inicjaly.bmp w formacie uint.8. Następnie przy pomocy funkcji stack z biblioteki NumPy zamieniamy tablicę na format RGB. Nadal mamy tam tylko 0 i 1. Zatem powstała funkcja konwertuj, która porówna wartości w tablicy i zamieni tak, że tam gdzie znajdzie wartości [1, 1, 1] zamieni na [255, 255, 255] czyli na biały. Następnie taką tablicę RGB dajemy jako argument do kolejnej funkcji, która w miejscu gdzie są narysowane inicjały zacznie rysować poziome pasy o ustalonej przez użytkownika grubości.





Obraz jpg Obraz png

Tak, widoczne są różnice między obrazami zapisanymi w formacie JPG i PNG. Główna przyczyna tych różnic wynika z różnicy w kompresji obu formatów.

- Format JPEG (JPG) jest formatem stratyowym, co oznacza, że kompresuje obraz przez usuwanie pewnych informacji, co może prowadzić do utraty jakości obrazu. Dlatego obrazy w formacie JPG są zazwyczaj mniejsze pod względem rozmiaru pliku, ale mogą wykazywać kompresję artefaktów, zwłaszcza przy niższych poziomach jakości.
- Format PNG jest formatem bezstratnym, co oznacza, że nie wprowadza kompresji stratowej i zachowuje pełną jakość obrazu. Obrazy w formacie PNG są zazwyczaj większe pod względem rozmiaru pliku, ale zachowują pełne szczegóły i jakość, dlatego są często wybierane do przechowywania obrazów, w których jakość jest ważniejsza niż rozmiar pliku.

Typ uint8 (unsigned 8-bit integer) w Pythonie i innych językach programowania oznacza, że liczby całkowite są reprezentowane w zakresie od 0 do 255. Jeśli podana wartość koloru przekracza 255, to typ uint8 "zawija się" i zaczyna od zera, co nazywa się przepełnieniem. Na przykład, 256 w typie uint8 jest reprezentowane jako 0. Jeśli wartość jest ujemna, typ uint8 nie obsługuje liczb ujemnych i znowu "zawija się", dlatego -1 w typie uint8 jest reprezentowane jako 255. To oznacza, że typ uint8 jest niemożliwy do reprezentowania wartości poza zakresem 0-255 i nie jest odpowiedni do reprezentacji kolorów lub innych danych, które wymagają wartości spoza tego zakresu. O to kod, który sprawdza pewne wartości w typie uint8:

```python
import numpy as np

tab = np.zeros((3), dtype=np.uint8)
x = 328
y = -1
z = -24
tab[0], tab[1], tab[2] = x, y, z
print(f'Wartość {x} w uint8 wynosi: {tab[0]}')
print(f'Wartość {y} w uint8 wynosi: {tab[1]}')
print(f'Wartość {z} w uint8 wynosi: {tab[2]}')
```

A to wyniki jakie zostały uzyskane:

```
Wartość 328 w uint8 wynosi: 72
Wartość -1 w uint8 wynosi: 255
Wartość -24 w uint8 wynosi: 232
```