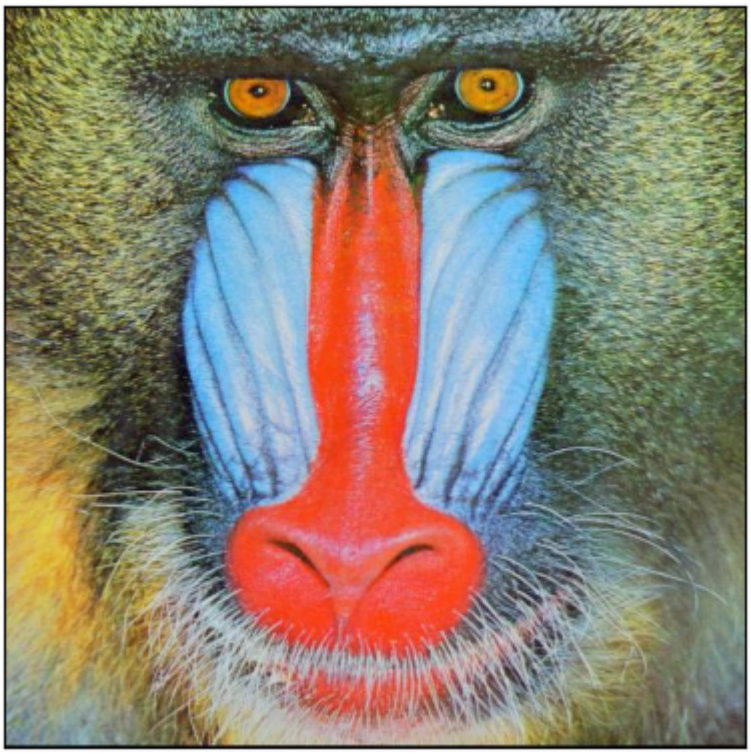
W ramach kursu będziemy wykorzystywać głównie bibliotekę OpenCV (Open Computer Vision Library). W trakcie niniejszego ćwiczenia zapoznamy się z podstawowymi funkcjonalnościami: wczytaniami, zapisywaniem, konwersją do inne przestrzeni barw. Zaznajomimy się również z Jupyter Notebook.

Wczytywanie obrazów z wykorzystaniem OpenCV.

1. Zaimportuj bibliotekę OpenCV:
   ```
   import cv2
   ```
2. Zaimportuj bibliotekę pyplot z matplotlib jako plt:
   ```
   from matplotlib import pyplot as plt
   ```
3. Wygodną opcją jest możliwość pobierania danych (obrazów, ale też np. baz danych) z sieci. Eliminuje to konieczność pobierania danych np. z Moodle oraz czyni uzupełniony Notebook "samowystarczalnym" - wystarczy go uruchomić.
   - zaimportuj bibliotekę os
     ```
     import os
     ```
   - zaimportuj bibliotekę request
     ```
     import requests
     ```
   - stwórz zmienne na adres pobieranego pliku (*url*) i samą jego nazwę (*fileName*)
   - sprawdź, czy taki plik istnieje (unikniemy wielokrotnego pobierania tych samych danych)
     ```
     if not os.path.exists(fileName):
     ```
   - pobierz plik: `r = requests.get(url + fileName, allow_redirects=True)`
   - zapisz na dysku: `open(fileName, 'wb').write(r.content)`
4. Wykorzystując polecenie *imread* wczytaj obraz *mandril.jpg*
5. Wyświetl obraz wykorzystując bibliotekę *pyplot*
   ```
   plt.imshow(I)
    plt.xticks([]), plt.yticks([])
    plt.show()
   ```
6. Jeśli ktoś nie wie jak powinien wyglądać mandryl, to proszę podglądnąć obraz na GitHub lub w sprawdzić w Interencie. Przyczyną problemu jest sposób w jaki funkcja `imread` z OpenCV odczytuje obraz (BGR, zamiast RGB). Obrazom kolorowym będzie poświęcone odrębne ćwiczenie, także nie będziemy w tym miejscu rozwijać tego wątku.
7. W celu poprawnego wyświetlenia należy dokonać konwersji koloru z BGR na RGB.
   ```
   I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
   ```
8. Dla porządku warto dodać, że w OpenCV do wyświetlania domyślnie wykorzystuje się funkcję `imshow`. Jednakże w przypadku Jupyter Notebook to rozwiązanie ma pewne wady i dlatego nie będziemy jej używać. Natomiast jeśli ktoś pracuje w "czystym" Python, to jest to na pewno równoważna funkcjonalność.

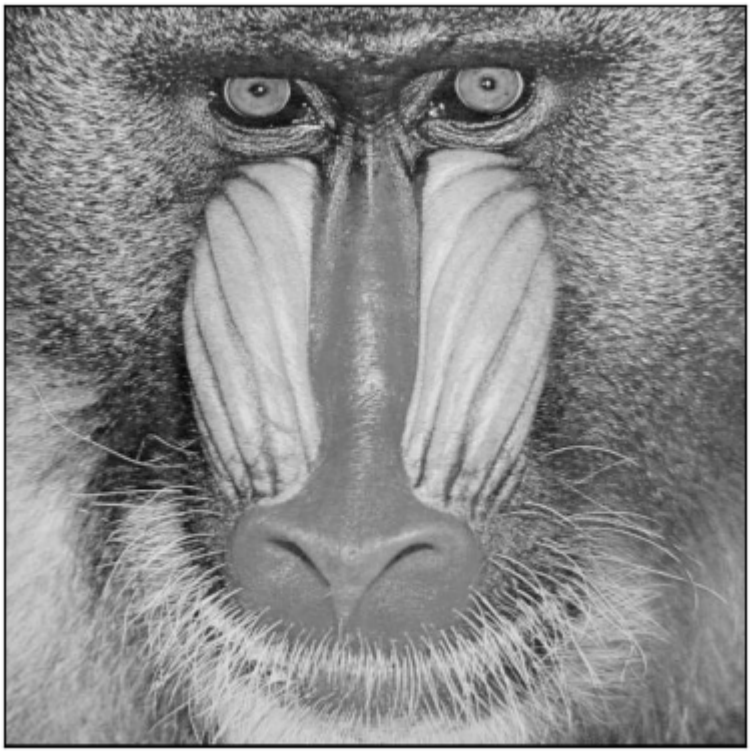
In [ ]:
```
import cv2
import os
```

```
from matplotlib import pyplot as plt
import requests

url = "https://raw.githubusercontent.com/vision-agh/poc_sw/master/01_Intro/"
image_name = "mandril"
filename = image_name + ".jpg"
if not os.path.exists(filename):
    r = requests.get(url + filename, allow_redirects=True)
    with open(filename, "wb") as image_file:
        image_file.write(r.content)

plt.xticks([])
plt.yticks([])
image = cv2.cvtColor(cv2.imread(filename), cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.show()
```



W praktycznych rozważaniach często analizuje się obrazy w odcieniach szarości (ang. *grayscale*). Do konwersji służy znana już funkcja `cvtColor` tylko tym razem z parametrem `cv2.COLOR_RGB2GRAY`. Proszę spróbować wyświetlić obraz. Może okazać się, że nadal jest kolorowy, choć inaczej. Dlaczego tak się dzieje zostanie obszerniej wyjaśnione pod koniec ćwiczenia. Na razie należy dodać polecenie `plt.gray()` przed `plt.show`.

In [ ]:
```
plt.xticks([])
plt.yticks([])
image_gs = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
plt.imshow(image_gs)
plt.gray()
plt.show()
```



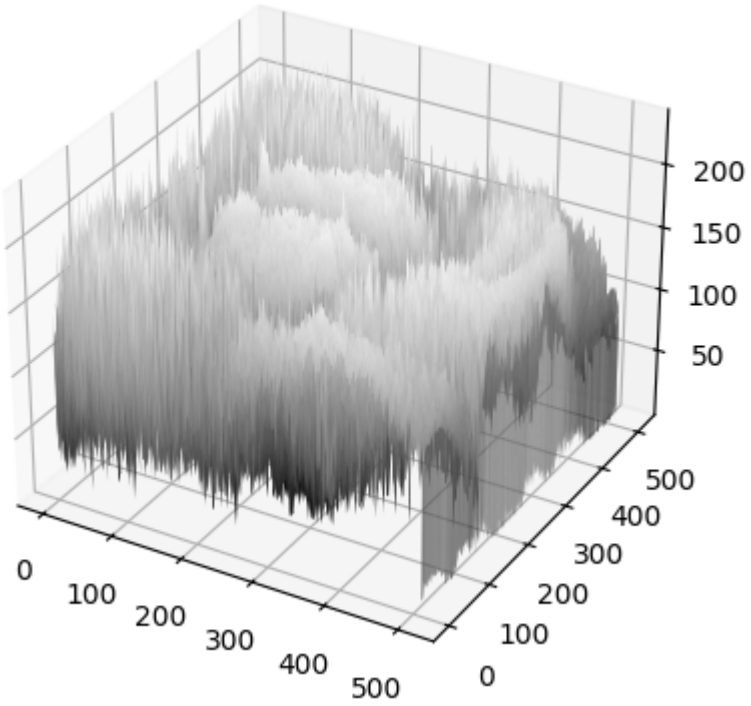
Czasami konieczne jest zapisanie przetworzonego obrazu. Służy do tego funkcja `cv2.imwrite`. Proszę zapisać szarą wersję mandryla w formacie *png*. Warto sprawdzić, czy obraz zapisał się poprawnie tj. otworzyć go w aplikacji do przeglądania obrazów.

In [ ]:
```
cv2.imwrite(image_name + ".gs.png", image_gs)
```

Out[ ]:
```
True
```

Obraz w skali szarości możemy traktować jako dwuwymiarową funkcję $L(x, y)$, gdzie x, y to współrzędne piksela, a $L(x, y)$ poziom jasności (najczęściej [0;255] - zapis na 8-bitach, typ *unsigned int*). Wyświetl obraz mandryl jako funkcję dwuwymiarową (uwaga - to chwilę się liczy):

In [ ]:
```
import numpy as np

# create grid
xx, yy = np.mgrid[0:image_gs.shape[0], 0:image_gs.shape[1]]

# create the figure
fig = plt.figure()
ax = fig.add_subplot(projection = '3d')
ax.plot_surface(xx, yy, image_gs ,rstride=1, cstride=1, cmap=plt.get_cmap('g

# show it
plt.show()
```



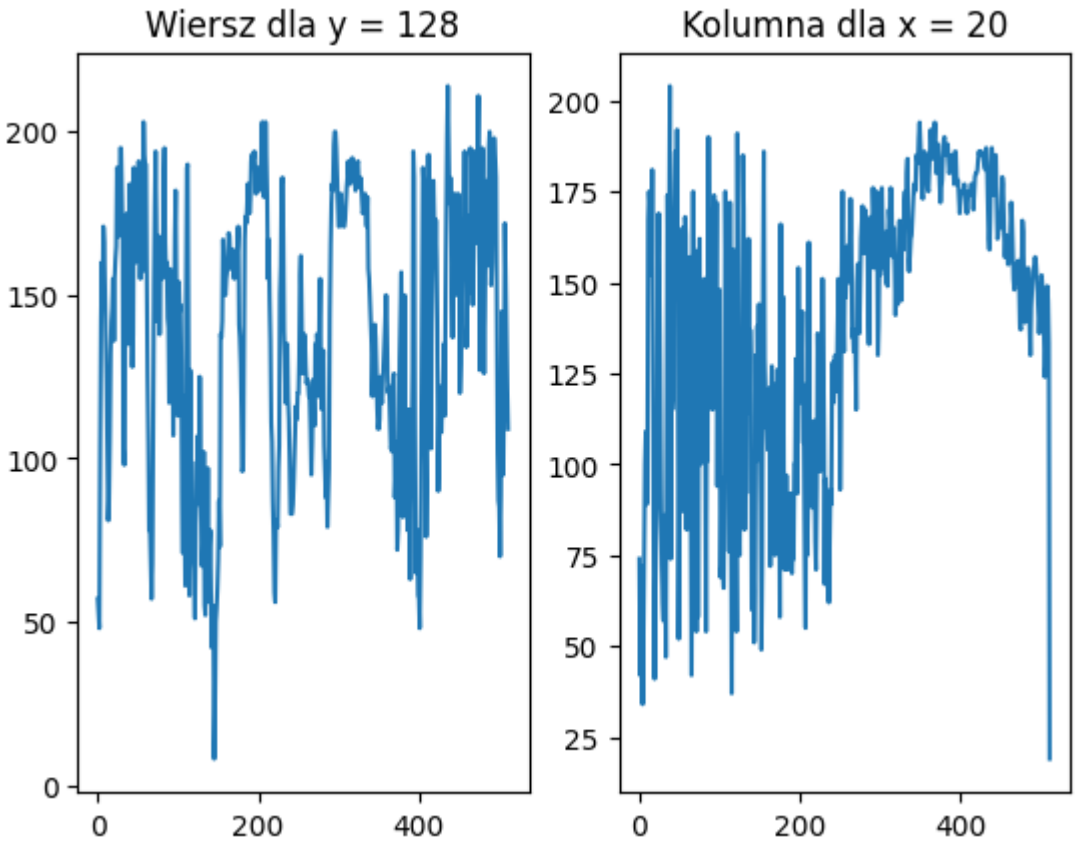
Komentarz:

- używamy biblioteki do obliczeń *numpy*,
- tworzymy siatkę punktów o rozmiarach takich jak obraz (*shape*),
- rysujemy wykres 3D

Podczas przetwarzania i analizy obrazów przydatne bywają "przekroje" przez obraz, czyli wartości funkcji $L(x, y)$ w przypadku gdy x lub y jest ustalone. Wykonaj jeden wybrany przekrój w x i y:

- po pierwsze należy pobrać dany wiersz lub kolumnę - np. `S = IG[10,:]` (tu jedenasty wiersz),
- po drugie stworzyć wykres złożony z dwóch subwykresów: `f, (ax1,ax2) = plt.subplots(1,2)`
- dla każdego z nich ustawić tytuł (np. `ax1.set_title('XXX')`) oraz treść `ax1.plot(S)`,
- na koniec całość wyświetlić `plt.show()`.

In [ ]:
```
x, y = 20, 128
image_row, image_col = image_gs[y, :], image_gs[:, x]
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.set_title(f"Wiersz dla y = {y}")
ax1.plot(image_row)
ax2.set_title(f"Kolumna dla x = {x}")
ax2.plot(image_col)
plt.show()
```



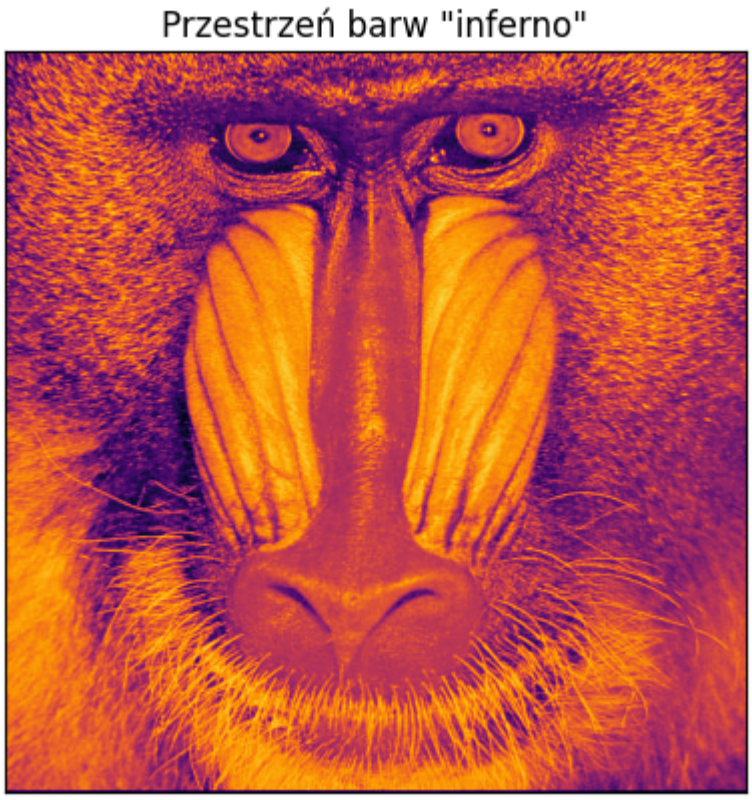
Obrazy indeksowane składają się z dwóch macierzy - obrazu oraz tzw. mapy kolorów. W macierzy obrazu zapisane są wartości poszczególnych pikseli. Macierz mapy kolorów ma rozmiar $m \times 3$ (m wierszy, w~każdym 3 wartości - składowe R,G,B). Podczas wyświetlania, na podstawie wartości piksela, odczytywany jest kolor z macierzy mapy kolorów. W ten sposób możliwe staje się "pokolorowanie" obrazu w skali szarości (stąd wcześniej szary mandryl był kolorowy). Wykorzystanie pseudokoloru nie wpływa na ilość informacji zawartej na obrazku, pomaga jedynie przedstawić go w bardziej czytelnej (dla człowieka) formie. Przykładowo można uzyskać poprawę kontrastu, co jest ważne przy analizie np. obrazów medycznych. Bardziej obszerny opis i spis dostępnym map można znaleźć w dokumentacji.

Wykorzystując dokumentację oraz polecenie `plt.imshow(IG, cmap=plt.get_cmap('XXX'))` proszę wyświetlić obraz madryl w 4 różnych mapach kolorów.

In [ ]:
```
def render_plot(image_gs, title, colormap):
  plt.xticks([])
  plt.yticks([])
  plt.title(title)
  plt.imshow(image_gs, cmap=colormap)
  plt.show()
```

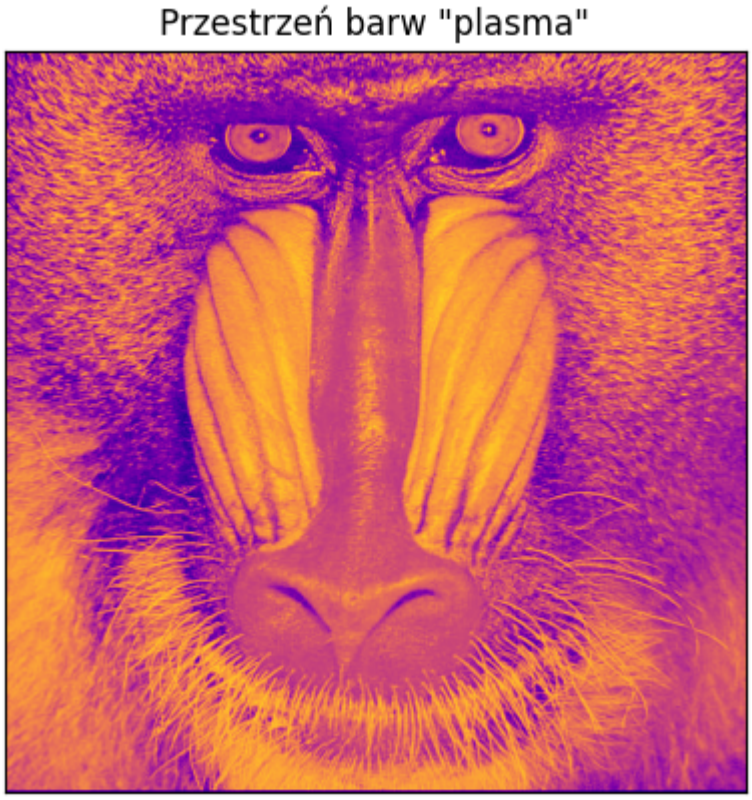
Inferno

In [ ]:
```
render_plot(image_gs, "Przestrzeń barw \"inferno\"", "inferno")
```



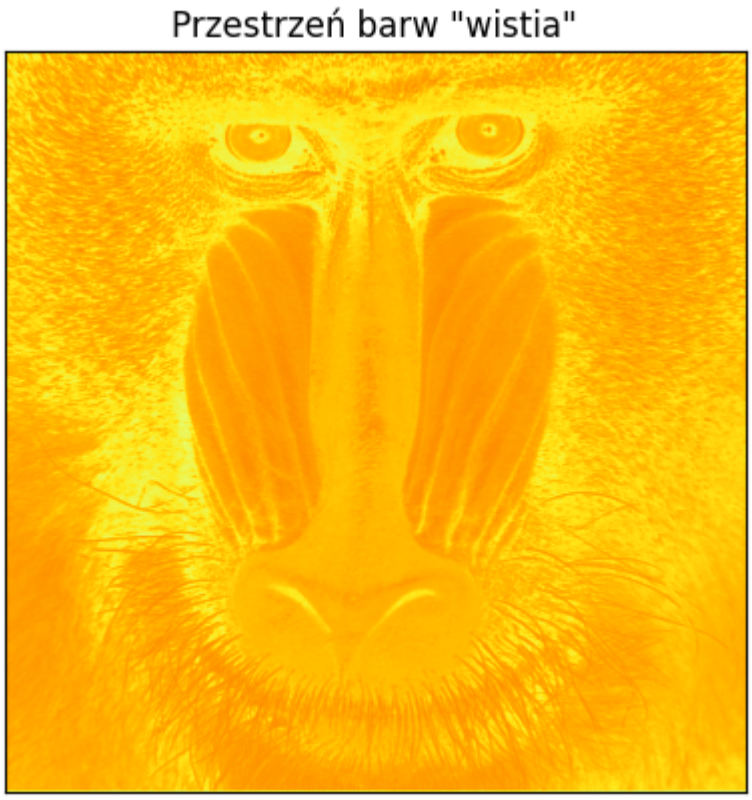
Plasma

In [ ]:
```
render_plot(image_gs, "Przestrzeń barw \"plasma\"", "plasma")
```



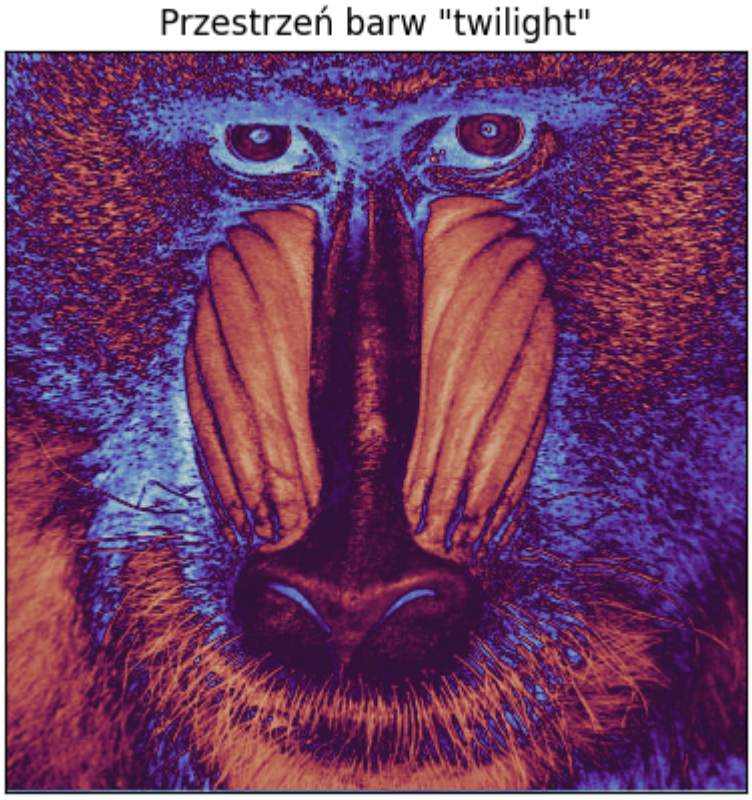
Wistia

In [ ]:
```
render_plot(image_gs, "Przestrzeń barw \"wistia\"", "Wistia")
```



Twilight

In [ ]:
```
render_plot(image_gs, "Przestrzeń barw \"twilight\"", "twilight")
```



Twilight shifted

In [ ]:
```
render_plot(image_gs, "Przestrzeń barw \"twilight shifted\"", "twilight_shi
```

Przestrzeń barw "twilight shifted"