

BÀI THỰC HÀNH 2

- *Tạo môi trường ảo Python*
- *Ảnh cơ bản với OpenCV*
- *Vẽ trên ảnh (Drawing on image)*

Nội dung:

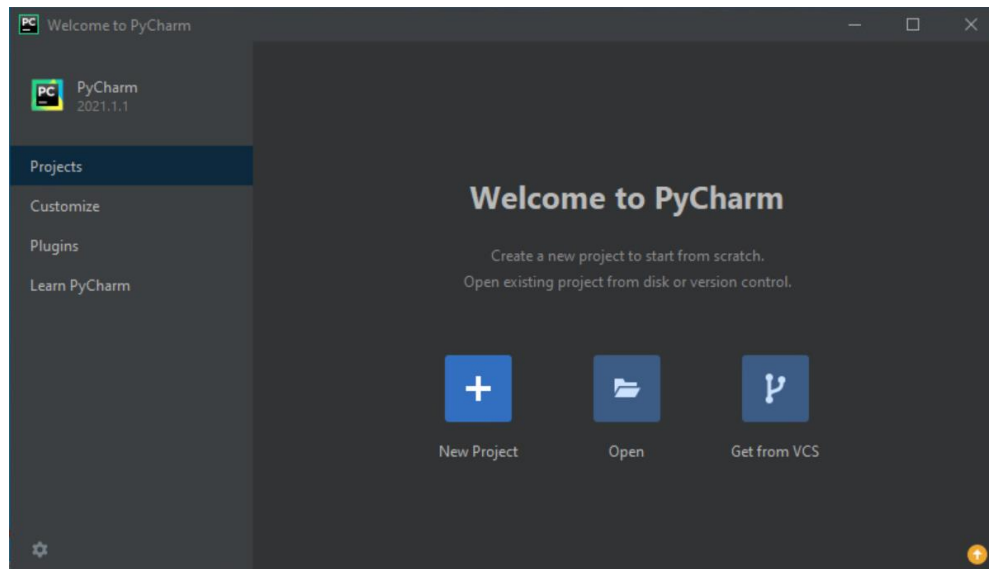
Thực hành trên Visual Studio Code (vscode) với môi trường ảo Python

- Tạo môi trường ảo (PyCharm) và sử dụng trong vscode trên máy ảo FIT.
- Thao tác ảnh cơ bản với OpenCV
- Vẽ trên ảnh

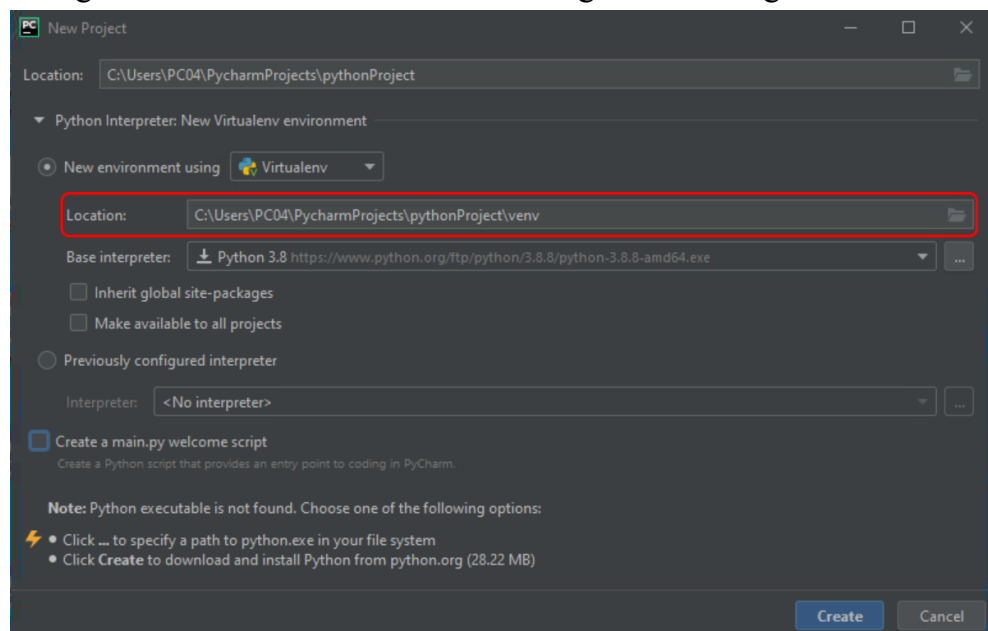
1. TẠO MÔI TRƯỜNG ẢO PYTHON

1.1. Tạo môi trường ảo với PyCharm (tài khoản user PCXX trên máy ảo FIT)

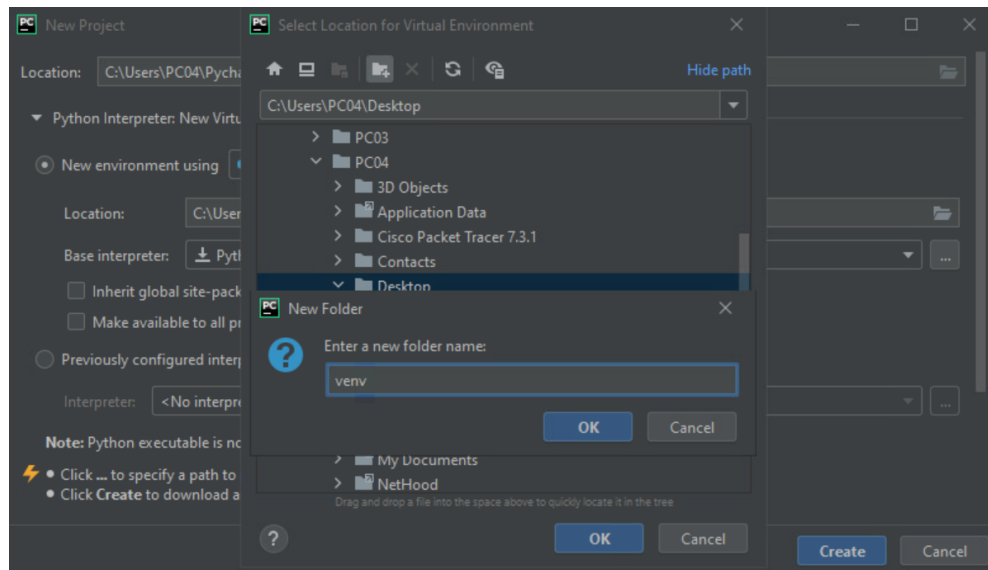
- Mở PyCharm Community trên máy ảo (chọn Don't send cho Data Sharing)
- Tạo Project mới (New project) trong PyCharm



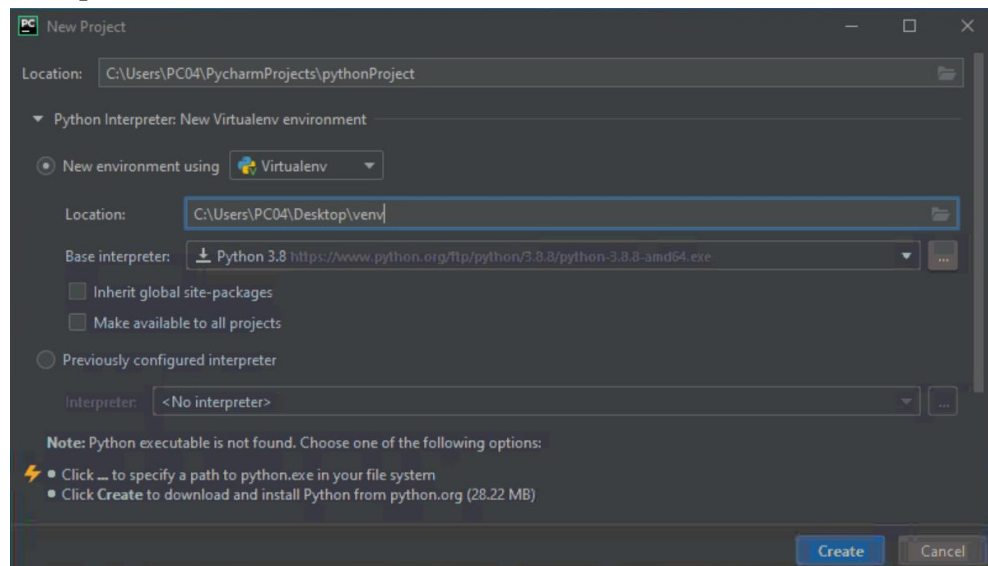
- Cần ghi nhận vị trí lưu trữ của môi trường ảo như trong hình sau:



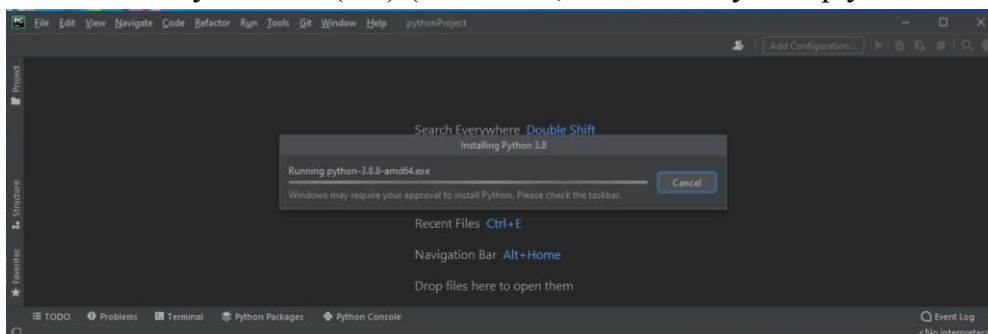
- Hoặc có thể đặt lại vị trí lưu trữ môi trường:



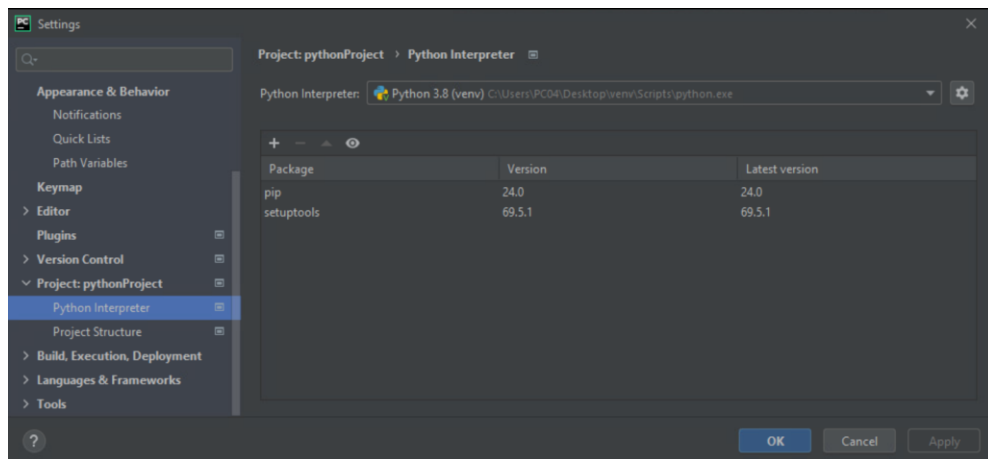
- Kết quả thu được:




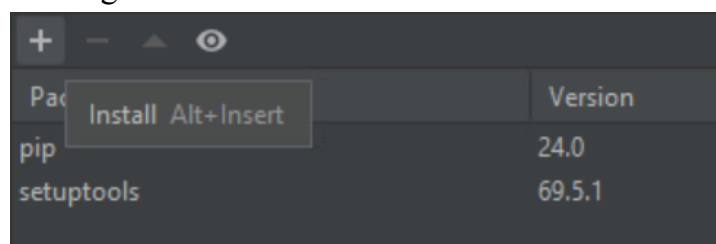
- Chọn Create để tạo Project mới trong PyCharm.
- Đợi cài đặt Python 3.8 (3.9) (Chọn Yes, khi cần chạy với quyền Admin)



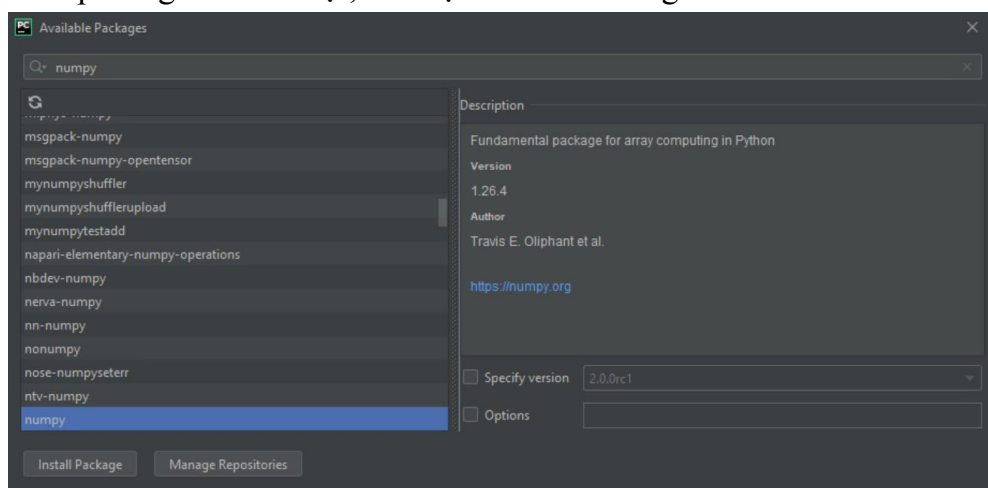
- Trong Settings của Project chọn Python Interpreter để thêm các package cần sử dụng như hình sau:



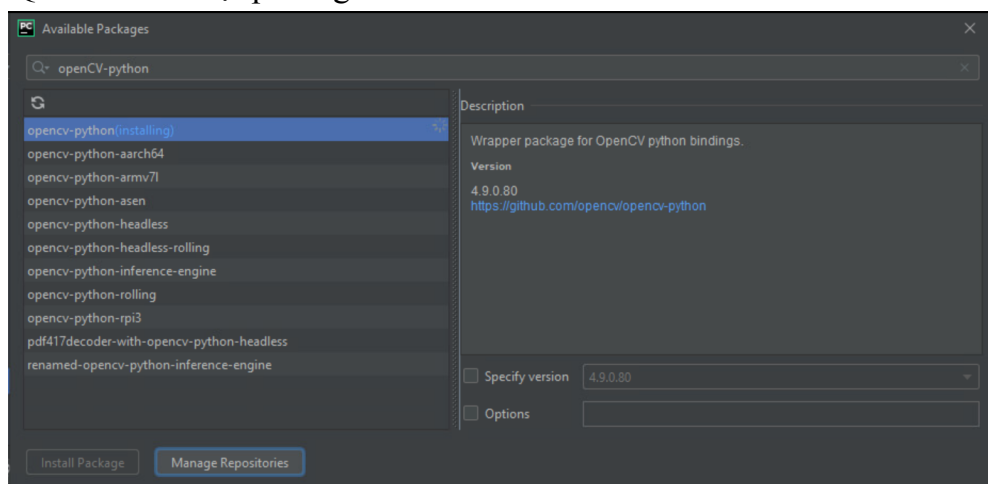
- Nhấn vào dấu  để lần lượt thêm numpy, matplotlib, openCV-python, và ipykernel vào môi trường:



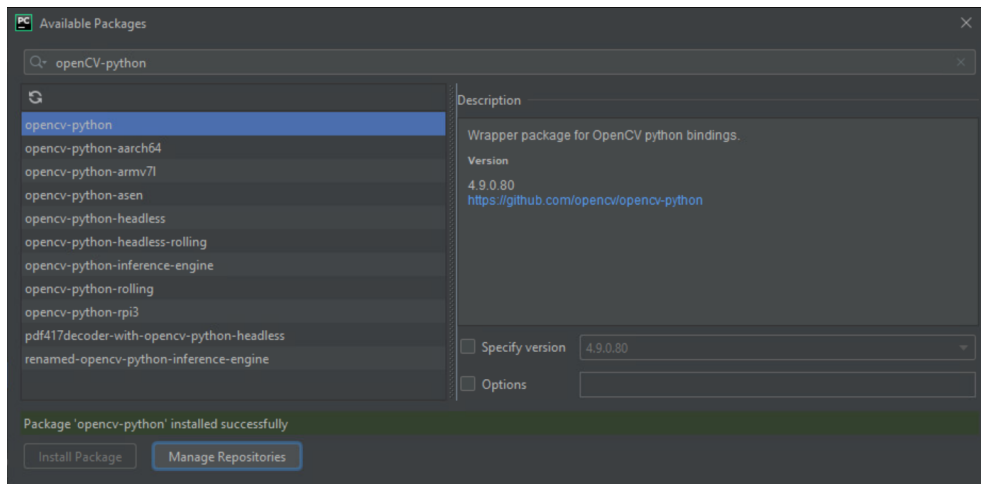
- Tìm package cần cài đặt, và chọn Install Package:



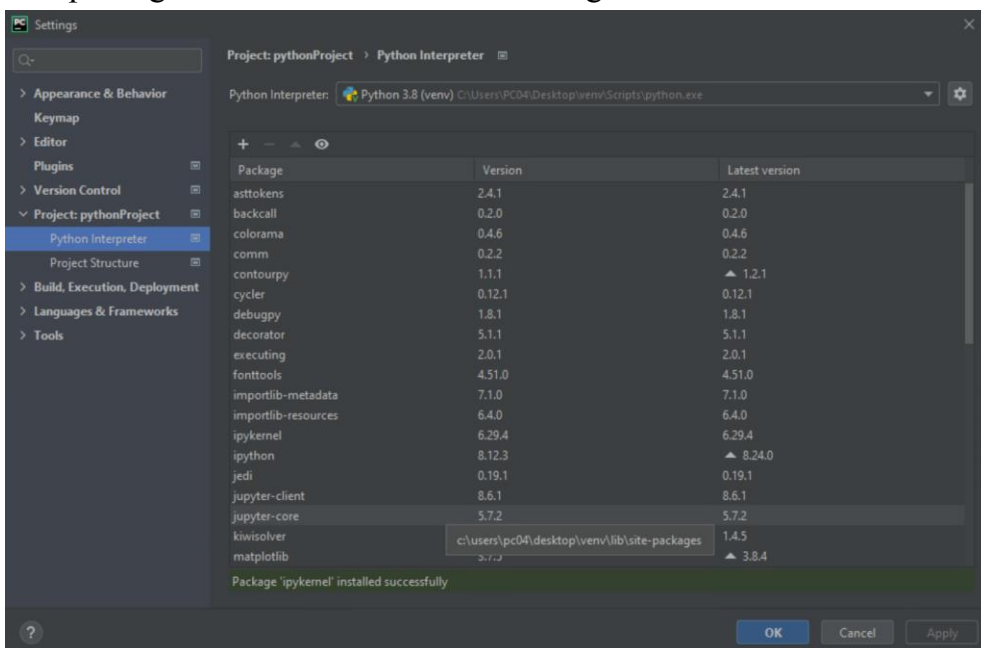
- Quá trình cài đặt package:



- Cài đặt package thành công:



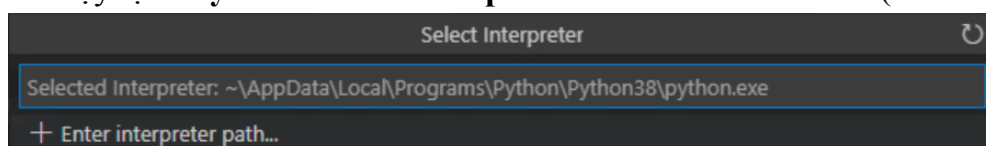
- Các package đã được thêm vào môi trường



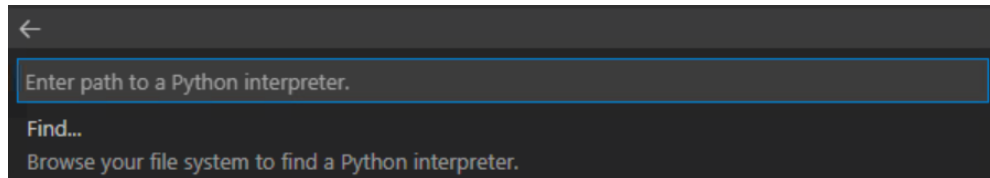
- Thoát Pycharm. Thư mục **venv** được sử dụng như môi trường ảo cho Python Interpreter trong Visual Studio Code (vscode).

1.2. Sử dụng môi trường ảo trong vscode (tài khoản user PCXX trên máy ảo FIT)

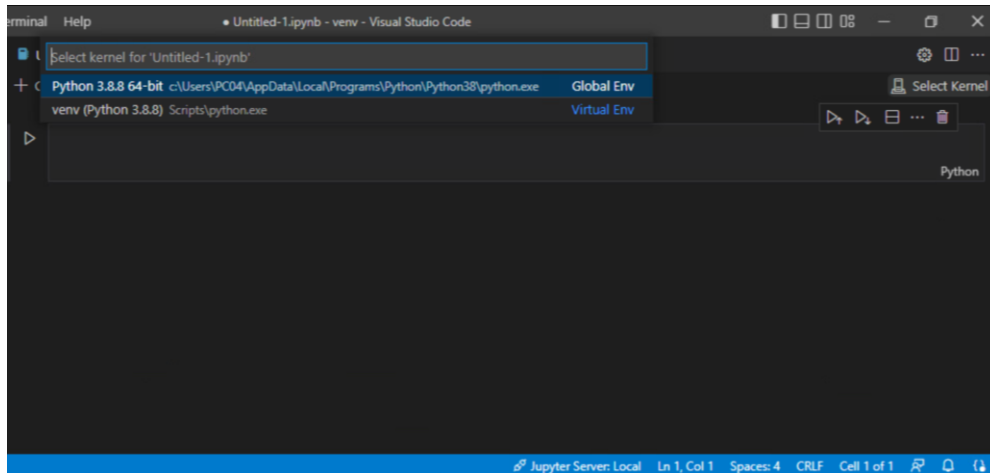
- Mở vscode trên máy ảo
- Chọn New File > Jupyter Notebook
- Cài đặt các Extension: Jupyter và Python (theo yêu cầu của vscode)
- Chọn Kernel (Select Kernel) là python từ môi trường đã tạo (thư mục **venv**) bằng cách chạy lệnh **Python: Select Interpreter** từ Command Palette (Ctrl+Shift+P).



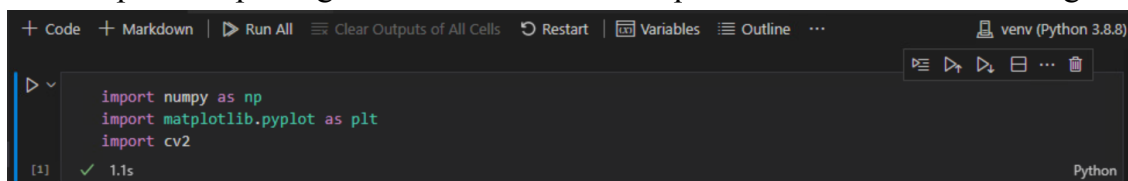
- Chọn Enter interpreter path...



- Chọn Find..., sau đó chọn đường dẫn đến “venv\Scripts\python.exe”
- Khởi động lại vscode; kết quả thu được:



- Import các package đã cài đặt để kiểm tra quá trình cấu hình môi trường ảo



2. THAO TÁC ẢNH CƠ BẢN VỚI OPENCV

2.1. Giới thiệu về OpenCV

OpenCV (Open Source Computer Vision) là một thư viện hàm lập trình tập trung chủ yếu vào thị giác máy tính thời gian thực (real-time computer vision).

Do Intel tạo ra vào năm 1999, được viết bằng ngôn ngữ C++. Bằng cách sử dụng Python bindings (OpenCV-Python), chúng ta có thể sử dụng trực tiếp ngôn ngữ lập trình Python cũng giống như các thư viện Python khác như Matplotlib hay NumPy.

2.2. Đọc ảnh với hàm `cv2.imread()`

OpenCV-Python là một thư viện Python bindings được thiết kế để giải quyết các bài toán thị giác máy tính. Hàm `cv2.imread()` cho phép nạp ảnh từ tập tin chỉ định. Nếu ảnh không thể đọc (do thiếu tập tin, quyền không đúng, hoặc định dạng không được hỗ trợ hoặc không hợp lệ) thì hàm sẽ trả về một ma trận rỗng.

❖ Cú pháp:

```
cv2.imread(filename, flag)
```

Trong đó:

filename: đường dẫn đến tập tin ảnh

flag: chỉ định cách thức ảnh được đọc

- **cv2.IMREAD_COLOR:** chỉ định sẽ nạp một ảnh màu. Mọi độ trong suốt của hình ảnh sẽ bị bỏ qua. Đây là cờ mặc định. Ngoài ra, chúng ta có thể truyền giá trị số nguyên 1 cho cờ.
- **cv2.IMREAD_GRAYSCALE:** chỉ định nạp một ảnh ở chế độ đa cấp xám. Ngoài ra, chúng ta có thể truyền giá trị số nguyên 0 cho cờ.
- **cv2.IMREAD_UNCHANGED:** chỉ định nạp vào một ảnh như vậy bao gồm cả kênh alpha. Ngoài ra chúng ta có thể truyền giá trị nguyên -1 cho cờ.

Giá trị trả về: hàm **cv2.imread()** trả về một NumPy array nếu ảnh được nạp thành công.

❖ Ví dụ:

- Import NumPy và Matplotlib

```
[1] import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
✓ 0.4s
```

- Import thư viện OpenCV

```
[5] import cv2
✓ 0.5s
```

- Đọc tập tin ảnh với hàm **cv2.imread()**:

```
[3] img=cv2.imread('images/puppy1.jpeg')
✓ 0.4s
```

- Kiểm tra kiểu dữ liệu của đối tượng biến **img**, nó được đọc trực tiếp như một NumPy array.

```
[4] type(img)
✓ 0.3s
... numpy.ndarray
```

- Chúng ta không cần thực hiện bước trung gian cho việc chuyển tập tin ảnh thành NumPy array, OpenCV đã làm chuyện đó tự động cho chúng ta bằng cách sử dụng **cv2.imread()**.

- Vấn đề quan trọng cần lưu ý khi sử dụng **cv2.imread()** đó là nếu đường dẫn hoặc tên tập tin bị sai (tập tin không tồn tại) nó vẫn không báo lỗi. Nhưng type sẽ cho kết quả là NoneType. Vì vậy tốt nhất nên kiểm tra kiểu của đối tượng biến nhận dữ liệu từ **cv2.imread()**

```
[6] img2=cv2.imread('wrong/path/doesnot/lfdfsdgrgr.jpg')
✓ 0.3s
```

```
▶ type(img2)
[7] ✓ 0.4s
... NoneType
```

- Về bản chất `img` là một NumPy Array chúng ta có thể xem dữ liệu của nó cũng như kiểm tra kích thước của nó như sau:

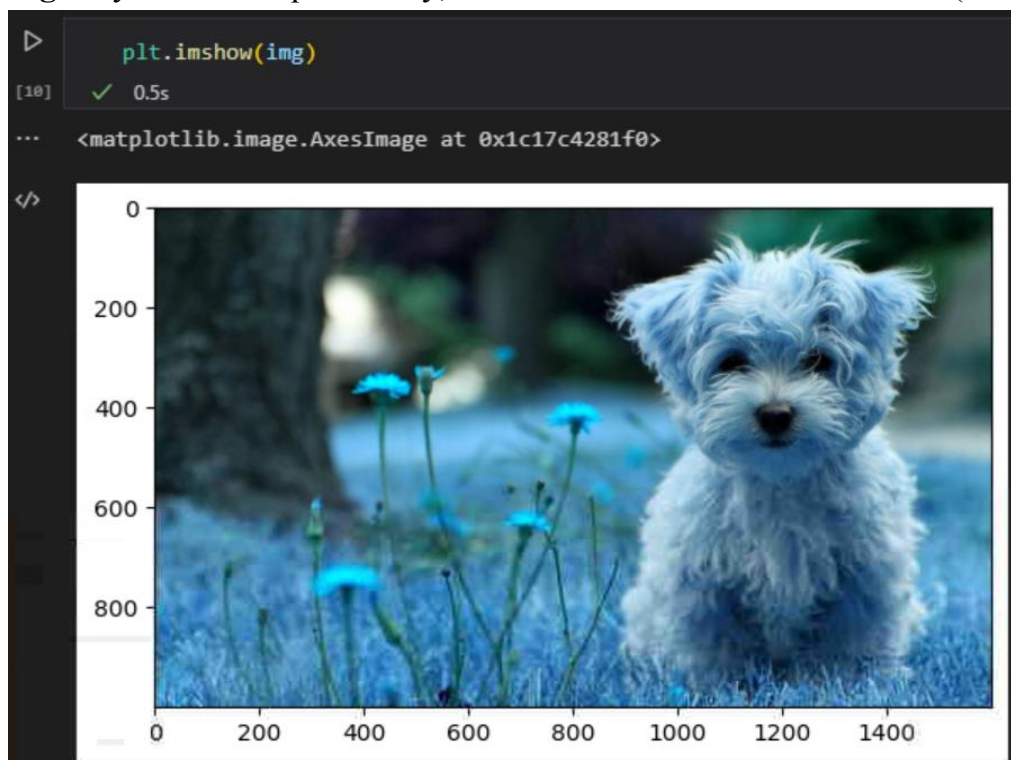
```
▶ img
[8] ✓ 0.3s
... Output exceeds the size limit. Open the full output data in a text editor
array([[ 13,  23,  23],
       [ 14,  24,  24],
       [ 14,  24,  24],
       ...,
       [  8,  56,  38],
```

- Kích thước của `img`

```
▶ img.shape
[9] ✓ 0.4s
... (1000, 1600, 3)
```

2.2.1. Vấn đề sử dụng hàm `imshow` của `matplotlib` để hiển thị ảnh đọc bởi hàm `imread` của `OpenCV`

- Chúng ta có thể sử dụng hàm `imshow` của `Matplotlib` để hiển thị dữ liệu ảnh từ biến `img`. Tuy nhiên cần quan sát kỹ, ảnh được hiển thị với sắc thái xanh (Blue) kỳ lạ.



Nguyên nhân của vấn đề này đó là `openCV` và `matplotlib` có thứ tự khác nhau giữa ba kênh màu Red, Green và Blue. Đó là sự khác biệt quan trọng cần ghi nhớ. Khi nghĩ về hình ảnh màu RGB, chúng được mong đợi các kênh Red, Green, Blue nên hiện diện

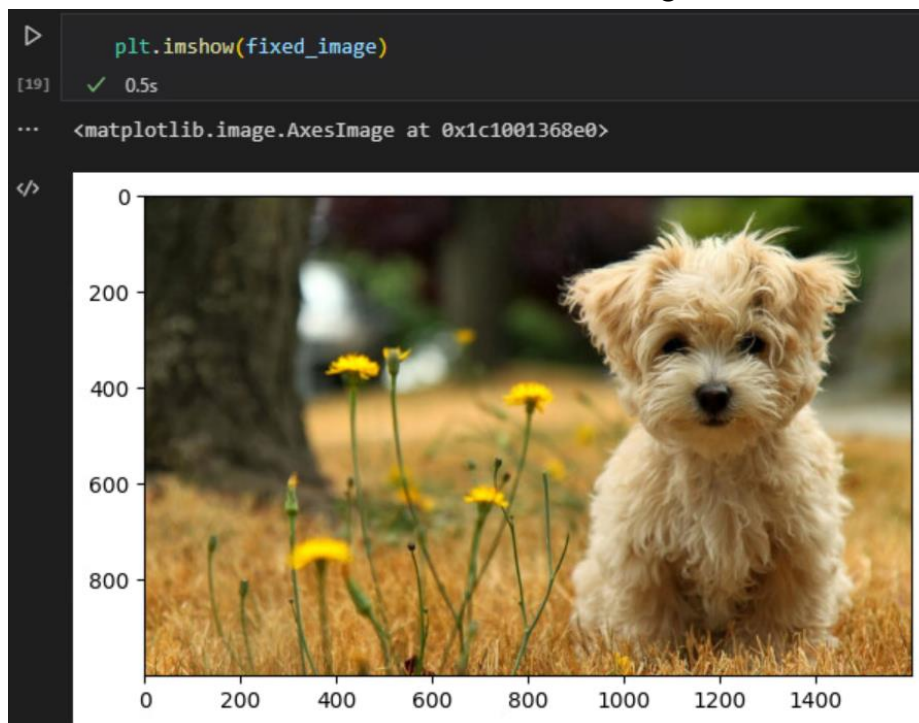
theo đúng thứ tự này, đó là Red, sau đó đến Green, và sau đó nữa là đến Blue. Tuy nhiên đối với OpenCV khi đọc một ảnh màu RGB, thứ tự này có sự khác biệt nhẹ. Đó là nó được đọc thành Blue, Green, Red.

Vậy việc gì thật sự đã xảy ra khi chúng ta đọc bức ảnh trên. Đó là do khi ảnh được đọc với `cv2.imread()` nó đã bị chuyển kênh và map với thứ tự kênh màu Blue, Green, Red. Trong khi Matplotlib hiển thị nó như một ảnh màu RGB. Hai thông số Blue và Red đã bị hoán đổi với nhau nên chúng ta có sự cố hiển thị như trên sau khi đọc ảnh với hàm `cv2.imread()`. Vậy làm thế nào để biến đổi ảnh từ Blue, Green, Red (BGR) thành Red, Green, Blue (RGB).

- OpenCV có hỗ trợ thực hiện biến đổi trên như sau:

```
fixed_image=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

- Hiển thị ảnh sau khi đã biến đổi từ BGR sang RGB:



Yêu cầu: Đọc ít nhất 5 ảnh khác nhau (tải ảnh về từ Internet, có định dạng JPG hoặc PNG, bao gồm các ảnh transparent background), xem kích thước ảnh đọc được, hiển thị ảnh, và ghi nhận kết quả.

2.2.2. Đọc ảnh gốc từ tập tin ảnh thành ảnh ở dạng đa cấp xám

- Trong trường hợp chúng ta muốn đọc vào một ảnh gốc như một ảnh đa cấp xám. OpenCV thật sự vẫn có thể làm điều đó bằng cách thêm vào color code cho hàm `cv2.imread()` như thông số thứ 2 của nó.

```
img_gray=cv2.imread('images/puppy1.jpeg', cv2.IMREAD_GRAYSCALE)
```

- Kiểm tra kích thước của `img_gray`; `img_gray` là một mảng 2 chiều có kích thước 1000x1600 (ảnh đơn sắc):


```
img_gray.shape
[21] ✓ 0.3s
... (1000, 1600)
```

2.2.3. *Hiển thị ảnh đa cấp xám được với hàm imshow của matplotlib*

- Dùng `plt.imshow()` để hiển thị ảnh đơn sắc **img_gray**, vấn đề màu sắc kỳ lạ vẫn xuất hiện. Nguyên nhân là do color mapping mặc định, giá trị tối hơn sẽ là các sắc thái của Blue, và giá trị sáng hơn sẽ được thể hiện ở sắc thái hơn màn hình hơi ngả vàng (Yellowish).



- Để khắc phục hiện tượng lạ trên, chúng ta có thể thêm thông số **cmap** cho hàm **imshow** của Matplotlib như sau:



Yêu cầu: thay đổi các giá trị cmap khác nhau cho hàm **imshow** của matplotlib (tham khảo Internet) và ghi nhận kết quả.

2.2.4. Hiện thị ảnh với hàm **cv2.imshow()**

Yêu cầu:

1. Thực hiện hiển thị các ảnh đã đọc bằng cách sử dụng hàm **cv2.imshow()** trong Jupyter Notebook. Hãy cho biết vấn đề gì xảy ra?
2. Trình bày nguyên nhân và giải pháp tối ưu cho vấn đề trên.

2.3. Thay đổi kích thước ảnh với hàm **cv2.resize()**

Thay đổi kích thước ảnh là nói đến việc lấy tỷ lệ ảnh. Việc lấy tỷ lệ có ích trong nhiều ứng dụng xử lý ảnh và thị giác máy vì nó giúp giảm số lượng pixel cần xử lý của một hình ảnh. Việc giảm số lượng pixel cần xử lý đem đến một số thuận lợi như giảm thời gian huấn luyện các mạng neural vì số lượng pixel trong ảnh càng nhiều thì số lượng nút (nodes) đầu vào sẽ càng nhiều, điều này làm gia tăng độ phức tạp của mô hình. Ngoài ra, việc thay đổi kích thước cũng giúp ích trong việc phóng to hình ảnh. Nhiều khi chúng ta cần thay đổi kích thước ảnh, tức là thu nhỏ hay tăng tỷ lệ ảnh để đáp ứng yêu cầu về kích thước. OpenCV cũng cấp cho chúng ta một số phương pháp nội suy để thay đổi kích thước hình ảnh.

❖ **Chọn phương pháp nội suy cho việc thay đổi kích thước ảnh:**

cv2.INTER_AREA: được sử dụng khi cần thu nhỏ hình ảnh, lấy mẫu lại dựa trên quan hệ vùng pixel;

cv2.INTER_CUBIC: cách này chậm nhưng hiệu quả hơn, nội suy hai chiều trên vùng lân cận 4×4 pixel;

cv2.INTER_LANCZOS4: nội suy Lanczos trên vùng lân cận 8×8;

cv2.INTER_NEAREST: nội suy lân cận gần nhất.

cv2.INTER_LINEAR: cách này chủ yếu được sử dụng khi cần thu phóng. Đây là phương pháp nội suy mặc định trong OpenCV.

❖ **Cú pháp:**

```
cv2.resize(source, dsize, dest, fx, fy, interpolation)
```

Trong đó:

source: ảnh đầu vào (kênh đơn sắc, số nguyên 8 bit hoặc số thực)

dsize: kích thước mảng đầu ra

dest: mảng đầu ra (có số chiều và kiểu dữ liệu giống với mảng đầu vào) (tùy chọn)

fx: hệ số tỷ lệ dọc theo trục ngang (tùy chọn)

fy: hệ số tỷ lệ dọc theo trục dọc (tùy chọn)

interpolation: một trong 3 phương pháp nội suy trên (tùy chọn)

❖ **Ví dụ:**

- Xem lại kích thước ảnh `fixed_imge`

```
fixed_image.shape
[24] ✓ 0.1s
... (1000, 1600, 3)
```

- Thay đổi kích thước ảnh về kích thước 1000×800 (số pixel chiều dọc \times số pixel chiều ngang):

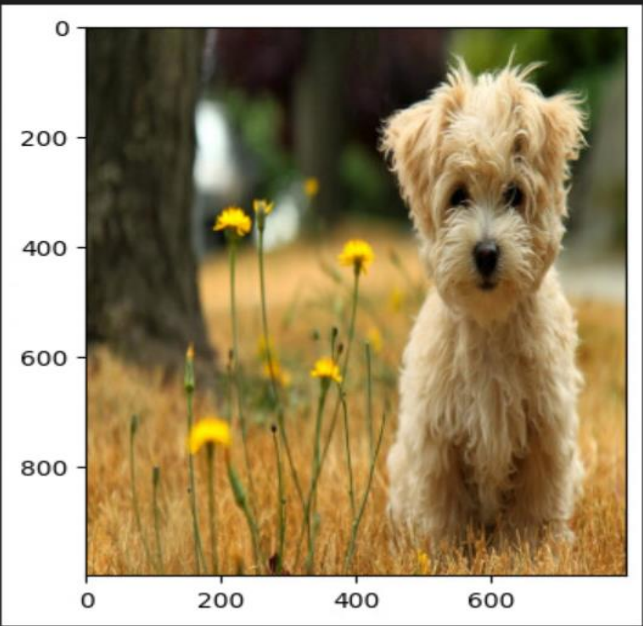
```
new_img=cv2.resize(fixed_image,(800, 1000))
[25] ✓ 0.4s
```

Lưu ý, kích thước mảng đầu ra sẽ được truyền theo thứ tự (số pixel chiều ngang, số pixel chiều dọc)

- Kết quả thu được:

```
new_img.shape
[26] ✓ 0.2s
... (1000, 800, 3)

plt.imshow(new_img)
[27] ✓ 0.3s
... <matplotlib.image.AxesImage at 0x1c100bed790>
</>
```

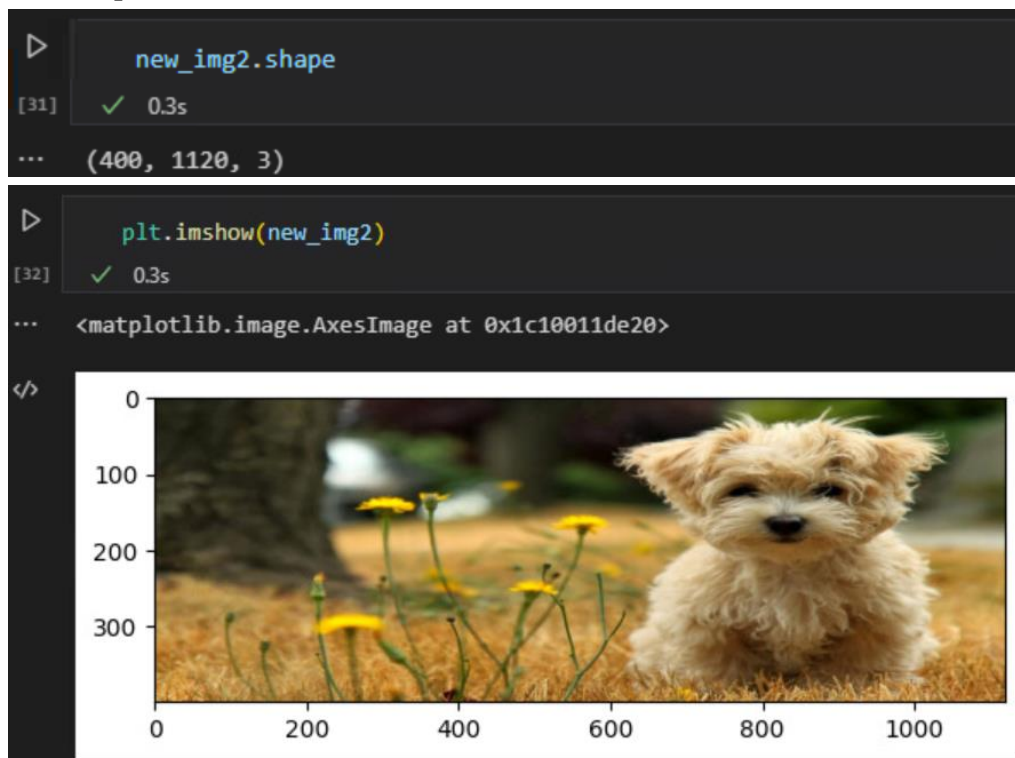


- Thay đổi kích thước theo ratio. Giả sử tỷ lệ (ratio) theo chiều ngang là 0.7 và theo chiều dọc là 0.4, chúng ta có thể thay đổi kích thước ảnh `fixed_image` như sau:

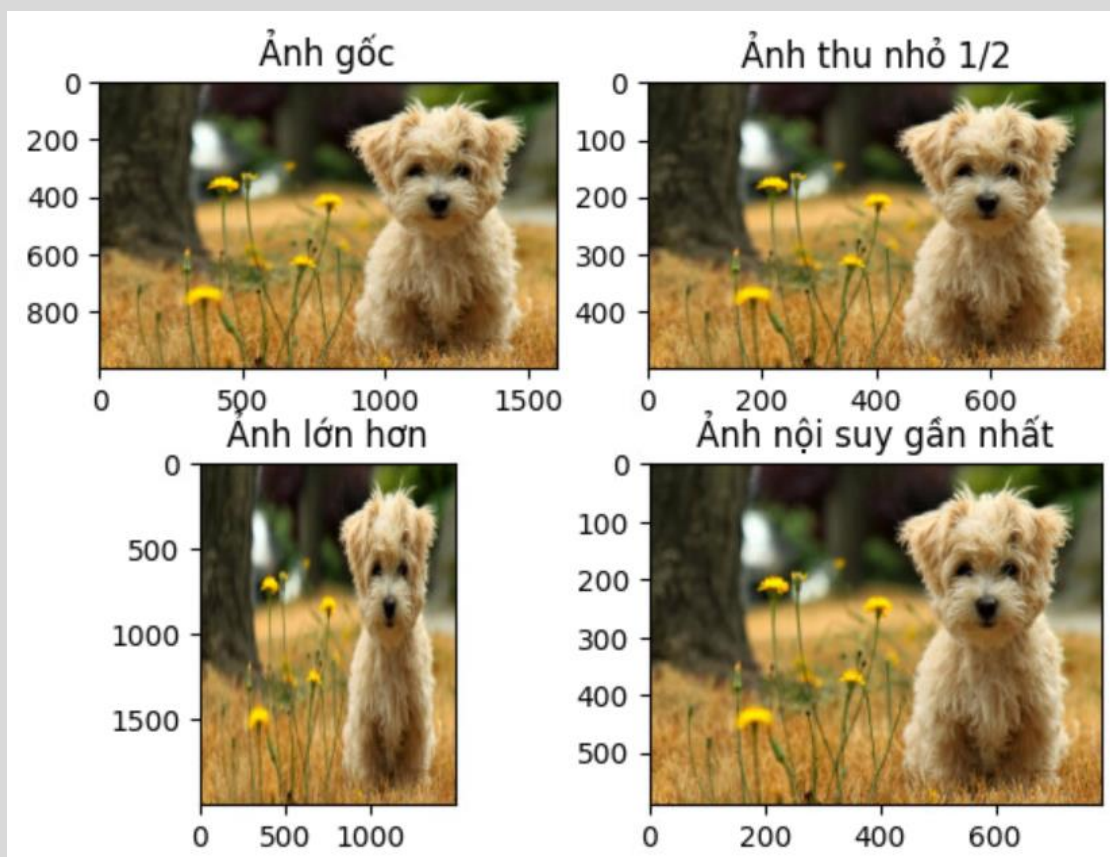
```
w_ratio=0.7 # chiều ngang
h_ratio=0.4 # chiều dọc
new_img2=cv2.resize(fixed_image, (0,0), fixed_image, w_ratio, h_ratio)
[30] ✓ 0.3s
```

Lưu ý, tham số thứ 3 của hàm `cv2.resize()`, `fixed_image`, có ý nghĩa mảng đầu ra có kích thước và số chiều giống với ảnh đầu vào (`fixed_image`)

- Kết quả thu được:



Yêu cầu: Viết đoạn chương trình thay đổi kích thước ảnh `fixed_image` để thu được kết quả như sau; với kích thước của “Ảnh lớn hơn” 2000×1500 , và kích thước của “Ảnh nội suy gần nhất” 590×785 . **Ràng buộc:** dùng vòng lặp cho việc tạo subplot và hiển thị ảnh.



2.4. Lật ảnh với hàm `cv2.flip()`

Lật ảnh là một quá trình đảo ngược ảnh theo chiều ngang hoặc chiều dọc. Việc lật ảnh hữu ích trong một số tác vụ xử lý ảnh khác nhau, chẳng hạn như tạo ảnh phản chiếu, lật ảnh để điều chỉnh hướng của nó, hoặc tạo ra dữ liệu huấn luyện bổ sung cho các thuật toán máy học.

Việc lật ảnh trong OpenCV là một xử lý đơn giản có thể được thực hiện bằng hàm `cv2.flip()`. Hàm `cv2.flip()` nhận vào 2 đối số chính: ảnh muốn lật và trục chúng ta muốn lật ảnh dọc theo. Chúng ta có thể lật ảnh qua trục ngang bằng cách truyền giá trị 0, và qua trục dọc bằng cách truyền giá trị 1 cho đối số thứ hai của hàm `cv2.flip()`. Giá trị -1 được sử dụng cho việc lật theo cả 2 trục.

❖ Cú pháp:

```
cv2.flip(src, flipCode[, dst] )
```

Trong đó:

src: ảnh (mảng) đầu vào

dst: mảng đầu ra có cùng kích thước và kiểu của src (tùy chọn)

flipCode: một cờ chỉ định cách thức lật mảng; 0 có nghĩa lật ảnh theo trục x (ngang), một giá trị dương (ví dụ 1) nghĩa là lật ảnh theo trục y (dọc). Giá trị âm (ví dụ -1) nghĩa là lật ảnh theo cả 2 trục.

Giá trị trả về: hàm trả về một ảnh là kết quả lật ảnh ban đầu.

❖ Ví dụ:

- Thực hiện lật ảnh theo trục y (dọc); hiển thị ảnh gốc và ảnh lật theo trục y:

```
# Lật ảnh theo trục y (dọc)
flipped_img1=cv2.flip(fixed_image,1)
plt.subplot(121)
plt.imshow(fixed_image)
plt.subplot(122)
plt.imshow(flipped_img1)
```

[76] ✓ 0.7s

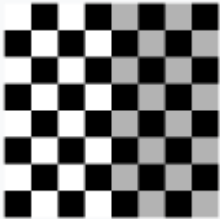

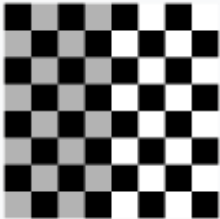
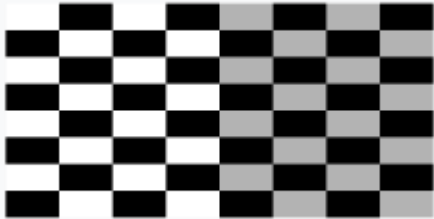
... <matplotlib.image.AxesImage at 0x1c104401220>





Yêu cầu: Thực hiện lật ảnh theo trục x và cả hai trục sử dụng hàm `cv2.flip()`, ghi nhận kết quả.

2.5. Biến đổi Affine 2D trong OpenCV

Yêu cầu: Trình bày các phép biến đổi Affine 2D cơ bản trên ảnh (dịch chuyển – pure translations, chiếu – reflection, lấy tỷ lệ theo hướng đã định – scaling in a given direction, xoay – rotation, kéo – shear) và viết đoạn code Python minh họa cho từng biến đổi này sử dụng các hàm dựng sẵn của OpenCV.

Tên biến đổi	Ví dụ
Identity (transform to original image)	
Translation	
Reflection	
Scale	

Tên biến đổi	Ví dụ
Rotate	 <p>where $\theta = \pi/6 = 30^\circ$</p>
Shear	

2.6. Ghi ảnh với hàm `cv2.imwrite()`

Hàm `cv2.imwrite()` được sử dụng cho việc lưu ảnh vào các thiết bị lưu trữ. Ảnh sẽ được lưu trữ theo định dạng (format) được chỉ định trong thư mục làm việc hiện thời.

❖ Cú pháp:

```
cv2.imwrite(filename, image)
```

Trong đó:

filename: là một chuỗi thể hiện tên tập tin. Tên tập tin phải bao gồm định dạng ảnh cần lưu trữ như *.jpg, *.png,...

image: là dữ liệu ảnh cần lưu trữ

Giá trị trả về: trả về true nếu ảnh được lưu trữ thành công.

Yêu cầu:

1. Ghi các dữ liệu ảnh đã thu được trong những nội dung trên thành các tập tin ảnh trong **thư mục được chỉ định trước** với định dạng *.jpg và *.png. Ghi nhận kết quả và nhận xét về chất lượng ảnh giữa 2 định dạng này. Gợi ý: thay đổi thư mục chỉ định ghi ảnh với module `os`.
2. Trình bày phương pháp thay đổi chất lượng ảnh được ghi ra với định dạng *.jpg. Viết code minh họa.

3. VẼ TRÊN ẢNH (DRAWING ON IMAGE)

3.1. Mục tiêu

Học vẽ các shape hình học khác nhau trên ảnh bằng cách sử dụng các hàm dựng sẵn của openCV

Các hàm vẽ trong openCV sẽ học bao gồm: `cv.line()`, `cv.circle()`, `cv.rectangle()`, `cv.ellipse()`, `cv.putText()`, ...

Lưu ý: Các hàm trên sẽ có một số đối số thông dụng như sau:

img: hình ảnh chúng ta muốn vẽ shape lên

color: màu sắc của shape. Đối với BGR, truyền nó như một bộ (tuple) các giá trị, ví dụ: (255,0,0) sẽ cho màu blue. Đối với ảnh đơn sắc (grayscale), chỉ cần truyền giá trị vô hướng (scalar value).

thickness: độ dày của đường thẳng hay hình tròn, Nếu truyền giá trị -1 cho các hình đóng như các vòng tròn, thì nó sẽ tô màu (fill) phần bên trong shape. Mặc định thickness có giá trị 1.

lineType: Kiểu của đường viền, liệu là đường liên kết 8 (8-connected line) hay là đường khử răng cưa (anti-aliases line)..., mặc định sẽ là đường liên kết 8. `cv2.LINE_AA` cho đường khử răng cưa, trong sẽ tốt hơn cho các vòng cung.

3.2. Các hàm vẽ shape trên ảnh

3.2.1. Tạo ảnh nền vẽ shape cơ bản

Chúng ta sẽ tạo ra một ảnh có kích thước $512 \times 512 \times 3$ màu đen để thực hiện vẽ các shape cơ bản trên đó. Lưu ý, có thể vẽ trực tiếp trên ảnh bất kỳ.

```
[1]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2

blank_img=np.zeros(shape=(512,512,3), dtype=np.int8)

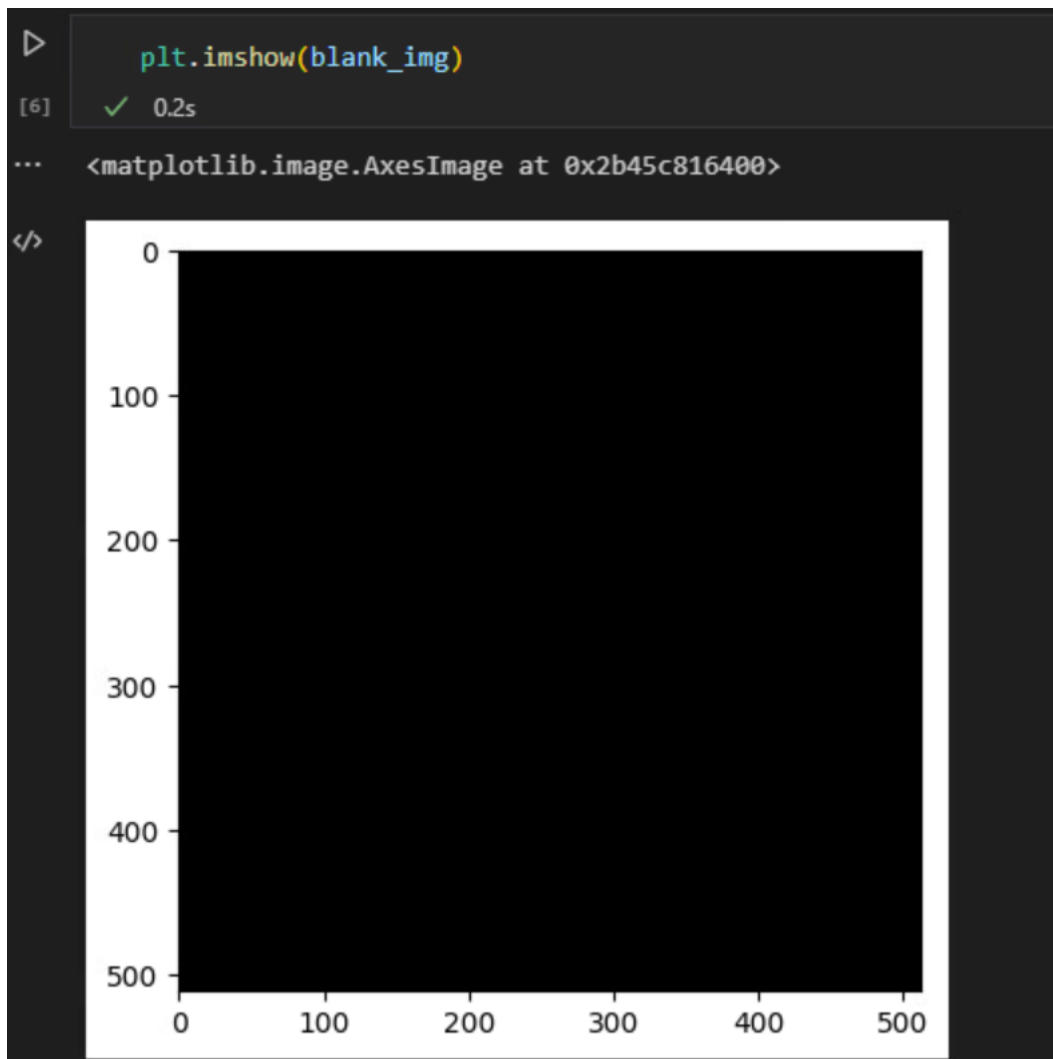
[1] ✓ 1.2s

[4]: type(blank_img)

[4] ✓ 0.2s
... numpy.ndarray

[5]: blank_img.shape

[5] ✓ 0.3s
... (512, 512, 3)
```



3.2.2. Vẽ đường thẳng với hàm `cv2.line()`

❖ Cú pháp:

```
cv2.line(img, start_point, end_point, color, thickness)
```

Trong đó:

start_point: tọa độ bắt đầu của đường thẳng; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

end_point: tọa độ kết thúc của đường thẳng; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

Giá trị trả về: trả về một hình ảnh

❖ Ví dụ:

- Tạo bản sao từ ảnh `blank_img`

```
blank_img_line=np.copy(blank_img)
```

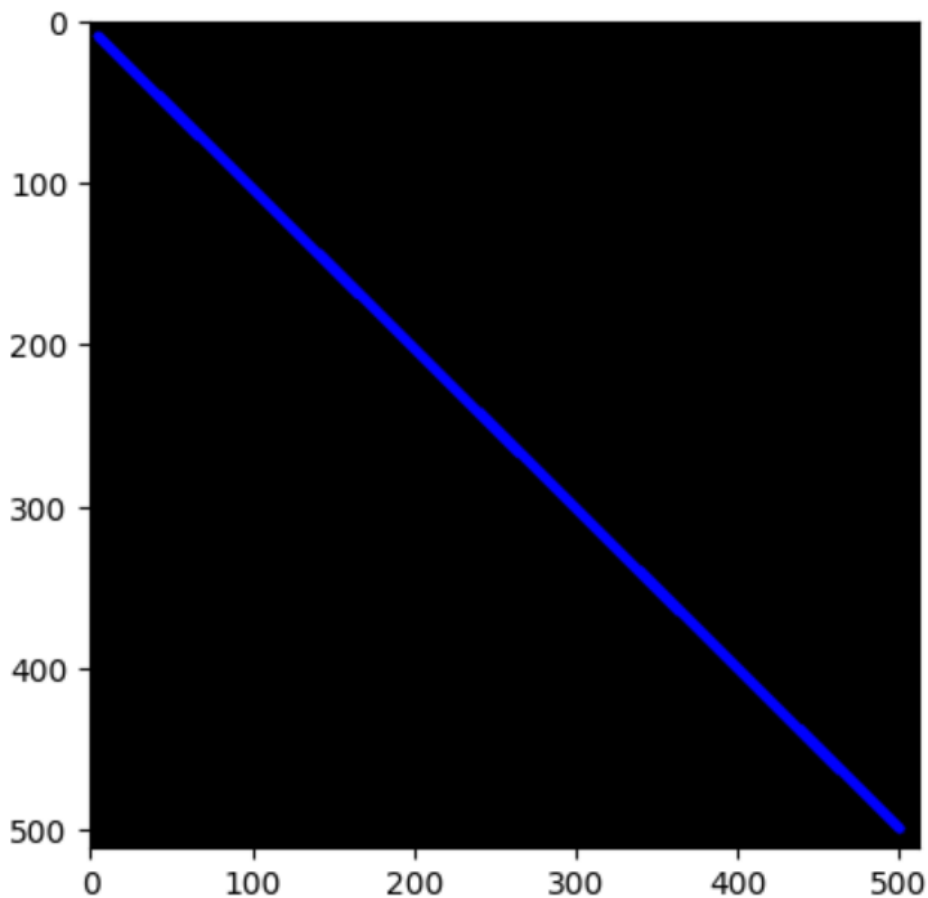
- Vẽ đường thẳng có màu xanh lam (blue) trên ảnh `blank_img_line` với tọa độ bắt đầu là (5,10) và tọa độ kết thúc là (500, 500), độ dày đường là 5.

```
cv2.line(blank_img_line, pt1=(5,10),pt2=(500, 500), color=(255, 0, 0), thickness=5)
```

- Kết quả thu được:

```
plt.imshow(cv2.cvtColor(np.float32(blank_img_line), cv2.COLOR_BGR2RGB))
```

[22] ✓ 0.1s



3.2.3. Vẽ hình chữ nhật với hàm `cv2.rectangle()`

❖ Cú pháp:

```
cv2.rectangle(img, start_point, end_point, color, thickness)
```

Trong đó:

start_point: tọa độ điểm góc trên bên trái hình chữ nhật; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

end_point: tọa độ điểm góc dưới bên phải hình chữ nhật; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

Giá trị trả về: trả về một hình ảnh

❖ Ví dụ:

- Tạo bản sao từ ảnh `blank_img`

```
blank_img_rectangle=np.copy(blank_img)
```

[23] ✓ 0.0s

- Vẽ hình chữ nhật rỗng, đường viền màu xanh lục (Green) với điểm bắt đầu là (120, 150) và điểm kết thúc là (250, 265), độ dày là 3

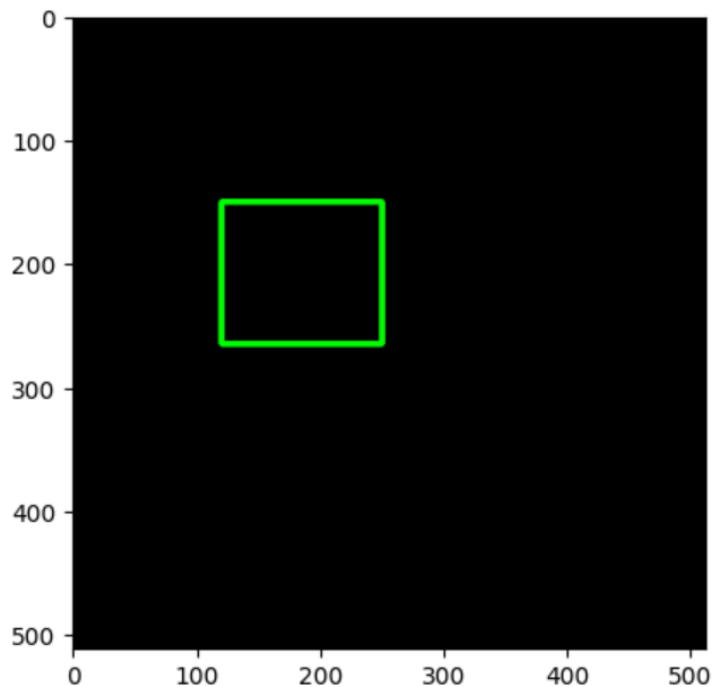
```
cv2.rectangle(blank_img_rectangle, (120, 150), (250, 265), (0, 255, 0), 3)
```

[31] ✓ 0.0s

- Kết quả thu được:

```
plt.imshow(cv2.cvtColor(np.float32(blank_img_rectangle), cv2.COLOR_BGR2RGB))
```

[30] ✓ 0.1s



3.2.4. Vẽ hình tròn với hàm `cv2.circle()`

❖ **Cú pháp:**

```
cv2.circle(image, center_coordinates, radius, color, thickness)
```

Trong đó:

center_coordinates: tọa độ tâm hình tròn; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

radius: bán kính đường tròn

Giá trị trả về: trả về một hình ảnh

❖ **Ví dụ:**

- Tạo bản sao từ ảnh `blank_img`

```
blank_img_circle=np.copy(blank_img)
```

[39] ✓ 0.0s

- Vẽ hình tròn rỗng, đường viền màu đỏ (Red) với tọa độ tâm là (340, 230), độ dày là 2.

```
cv2.circle(blank_img_circle, center=(340, 230), radius=40, color=(0, 0, 255), thickness=2)
```

[40] ✓ 0.0s

- Kết quả thu được:



3.2.5. Vẽ hình ellipse với hàm `cv2.ellipse()`

❖ Cú pháp:

```
cv2.ellipse(image, centerCoordinates, axesLength, angle,
startAngle, endAngle, color [, thickness[, lineType[, shift]]])
```

Trong đó:

centerCoordinates: tọa độ tâm hình ellipse; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

axesLength: là bộ giá trị tương ứng với độ dài trục lớn và trục nhỏ của ellipse, (độ dài trục lớn, độ dài trục nhỏ).

angle: ellipse quay góc bao nhiêu độ

startAngle: góc bắt đầu của cung ellipse, tính bằng độ

endAngle: góc kết thúc của cung ellipse, tính bằng độ

Giá trị trả về: trả về một hình ảnh

❖ Ví dụ:

- Tạo bản sao từ ảnh `blank_img`

```
blank_img_ellipse=np.copy(blank_img)
```

- Vẽ hình ellipse, đường viền màu xanh lơ (Cyan) với tọa độ tâm là (256, 256), quay 45 độ, góc bắt đầu của cung ellipse là 0, góc kết thúc của cung ellipse là 360, độ dày là 3.

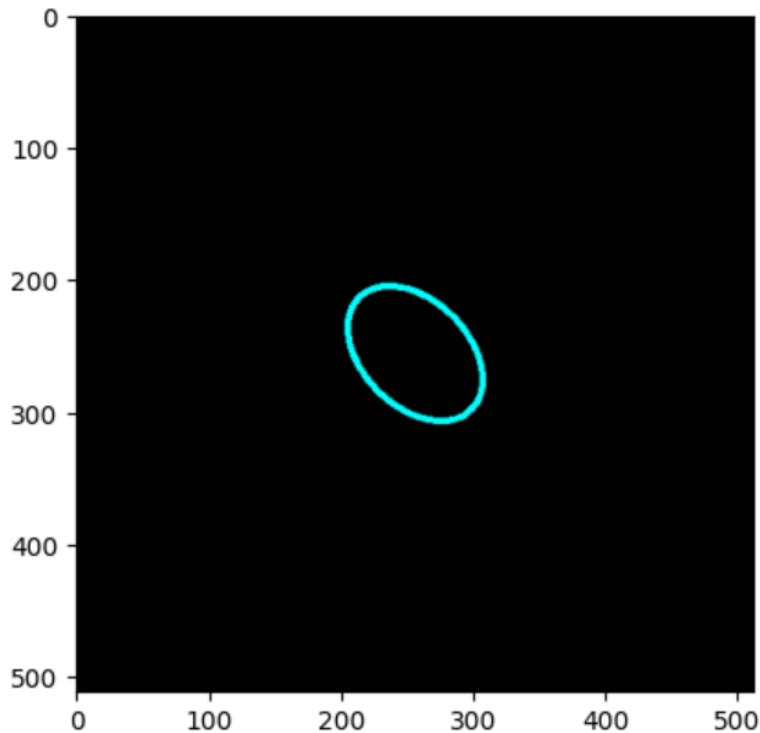
```
cv2.ellipse(blank_img_ellipse, (256, 256), (60, 40), 45.0, 0.0, 360.0, (255, 255, 0), 3)
```

[48] ✓ 0.0s

- Kết quả thu được:

```
plt.imshow(cv2.cvtColor(np.float32(blank_img_ellipse), cv2.COLOR_BGR2RGB))
```

[50] ✓ 0.1s



3.2.6. Đưa văn bản vào hình với hàm `cv2.putText()`

❖ **Cú pháp:**

```
cv2.putText(image, text, org, font, fontScale, color[,  
            thickness[, lineType[, bottomLeftOrigin]])
```

Trong đó:

Text: nội dung văn bản cần đưa vào hình

org: tọa độ góc trên bên trái của văn bản trong hình; tọa độ được thể hiện bởi một bộ 2 giá trị nghĩa là (giá trị tọa độ X, giá trị tọa độ Y)

font: chỉ định loại phông chữ; một số loại phông chữ có thể sử dụng như FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN,...

fontScale: Hệ số tỷ lệ phông chữ được nhân với kích thước cơ sở của phông chữ cụ thể.

bottomLeftOrigin: Khi giá trị này đúng, nguồn gốc dữ liệu hình ảnh nằm ở góc dưới bên trái. Nếu không thì nó nằm ở góc trên bên trái (Tùy chọn)

Giá trị trả về: trả về một hình ảnh

❖ **Ví dụ:**

- Tạo bản sao từ ảnh `blank_img`

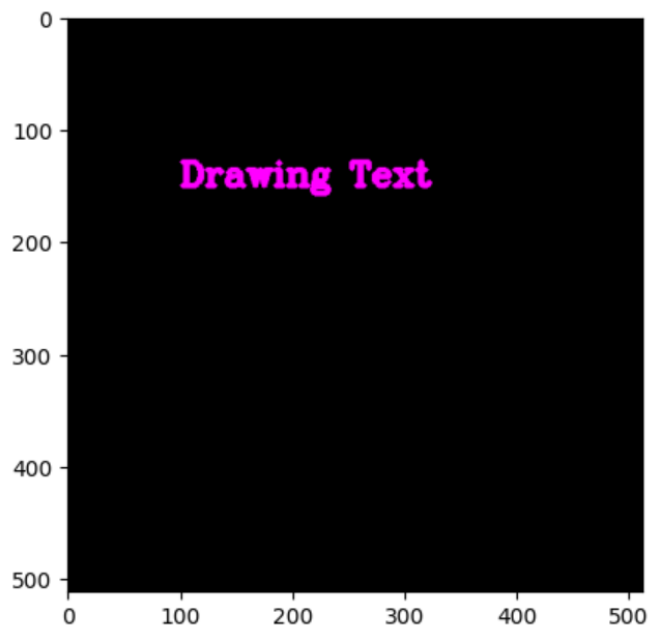
```
blank_img_Text=np.copy(blank_img)
[51] ✓ 0.0s
```

- Đưa văn bản vào hình.

```
cv2.putText(blank_img_Text,'Drawing Text', (100, 150), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 255), 3)
[58] ✓ 0.0s
```

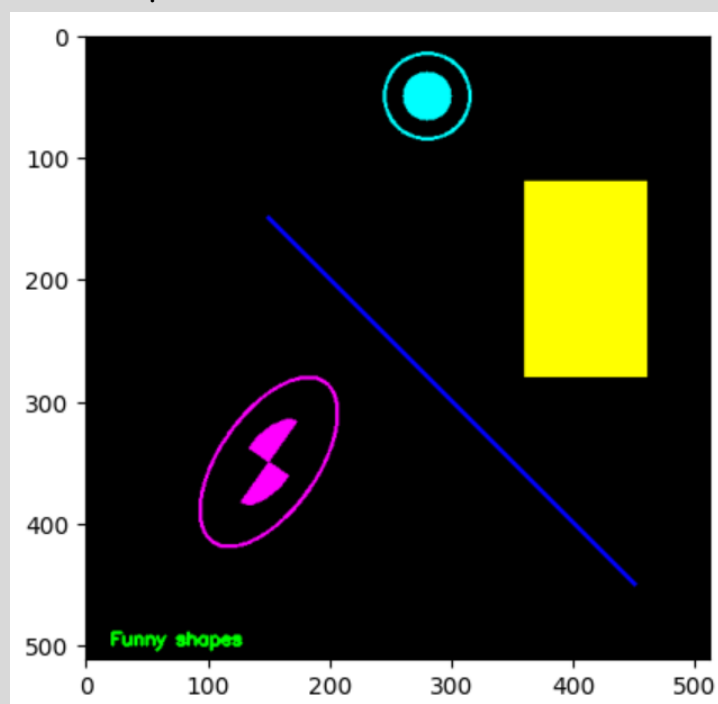
- Kết quả thu được:

```
plt.imshow(cv2.cvtColor(np.float32(blank_img_Text),cv2.COLOR_BGR2RGB))
[59] ✓ 0.1s
```



Yêu cầu:

- 1. Thay đổi từng giá trị cho từng đối số trong các hàm drawing ở trên, ghi nhận kết quả.*
- 2. Viết đoạn code để thu được hình ảnh như sau:*



3.2.7. Vẽ polygon (đa giác) với hàm `cv2.polylines()`

❖ Cú pháp:

```
cv2.polylines(image, [pts], isClosed, color, thickness)
```

Trong đó:

pts: mảng các cung đa giác

isClosed: Cờ cho biết có đóng đa giác hay không. Nếu có thì sẽ có đường nối giữa đỉnh cuối cùng với đỉnh đầu tiên của đa giác.

Giá trị trả về: trả về một hình ảnh

❖ Ví dụ:

- Tạo bản sao từ ảnh `blank_img`

```
blank_img_polygon=np.copy(blank_img)
```

[15] ✓ 0.0s

- Tạo tập tọa độ các đỉnh của polygon là một NumPy array

```
vertices=np.array([[25, 70], [25, 160],  
                  [110, 200], [200, 160],  
                  [200, 70], [110, 20]],  
                  dtype=np.int32)
```

[20] ✓ 0.0s

- Kiểm tra kết quả tạo tập đỉnh polygon (vertices) và kích thước của nó

```
vertices
```

[22] ✓ 0.0s

```
... array([[ 25,  70],  
          [ 25, 160],  
          [110, 200],  
          [200, 160],  
          [200,  70],  
          [110,  20]])
```

```
vertices.shape
```

[21] ✓ 0.0s

```
... (6, 2)
```

- Mặc dù tập đỉnh có vẻ đã đúng định dạng (tập hợp tọa độ các đỉnh), tuy nhiên đối với OpenCV các đỉnh này nên ở trong không gian 3D. Vì vậy, chúng ta sẽ thực hiện chuyển kích thước vertices về dạng 3D với hàm `reshape` của NumPy như sau:

```
vertices=vertices.reshape((-1,1,2))
```

[24] ✓ 0.0s

Trong câu lệnh trên, giá trị -1 trong bộ giá trị mang ý nghĩa giá trị của chiều thứ nhất sẽ được suy ra từ độ dài của mảng và các kích thước còn lại.

- Kiểm tra lại lần nữa kích thước của vertices sau khi reshape:

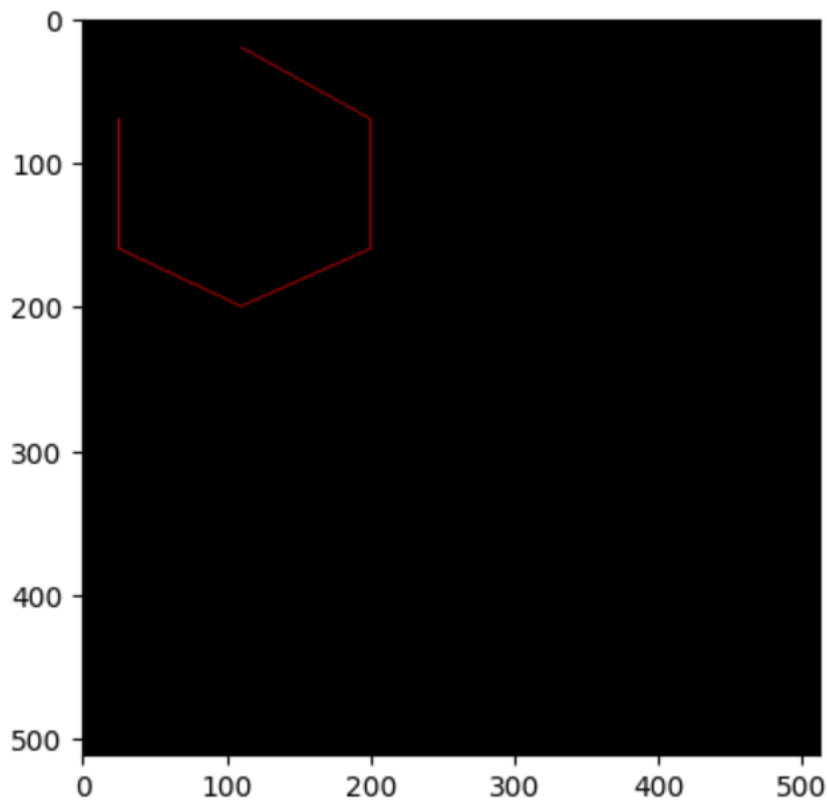
```
▶ ▼ vertices.shape  
[25] ✓ 0.0s  
... (6, 1, 2)
```

- Vẽ polygon với các đỉnh đã tạo, khi không đóng đa giác:

```
▶ ▼ cv2.polylines(blank_img_polygon,[vertices],isClosed=0,color=(0,0,255))  
[29] ✓ 0.0s
```

- Kết quả thu được:

```
▶ ▼ plt.imshow(cv2.cvtColor(np.float32(blank_img_polygon),cv2.COLOR_BGR2RGB))  
[19] ✓ 0.1s
```

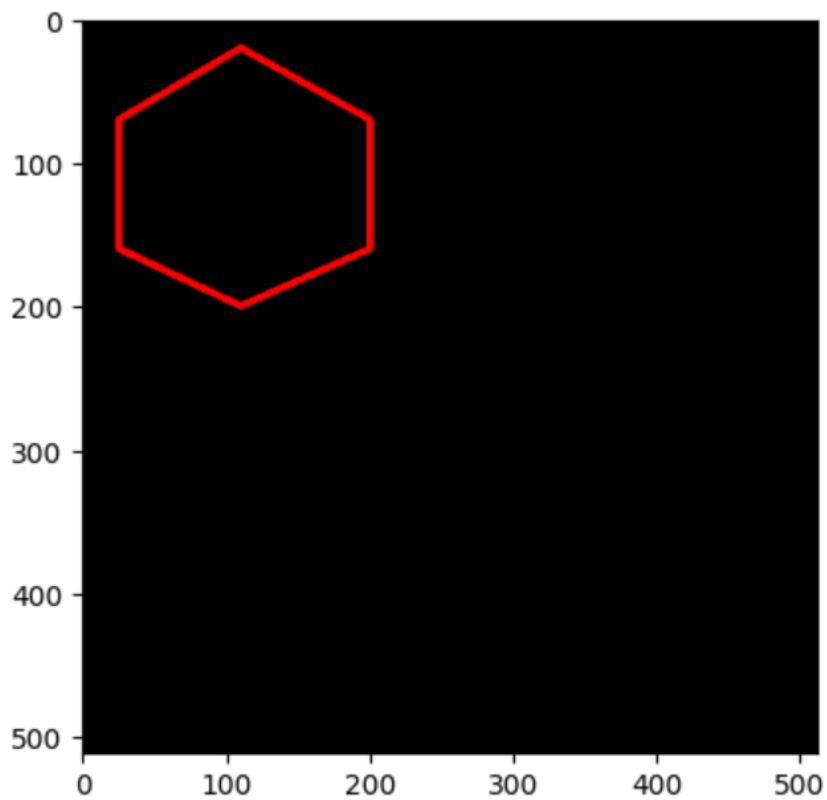


- Vẽ polygon với các đỉnh đã tạo, khi đóng đa giác:

```
▶ ▼ cv2.polylines(blank_img_polygon,[vertices],isClosed=1,color=(0,0,255),thickness=3)  
[29] ✓ 0.0s
```

- Kết quả thu được:

```
▶ ▼ plt.imshow(cv2.cvtColor(np.float32(blank_img_polygon),cv2.COLOR_BGR2RGB))  
[33] ✓ 0.1s
```



Yêu cầu:

Viết đoạn code để thu được hình ảnh như sau, biết các đỉnh polygon lần lượt là:
[256, 128], [392, 235], [335, 384], [177, 384], [120, 235]

