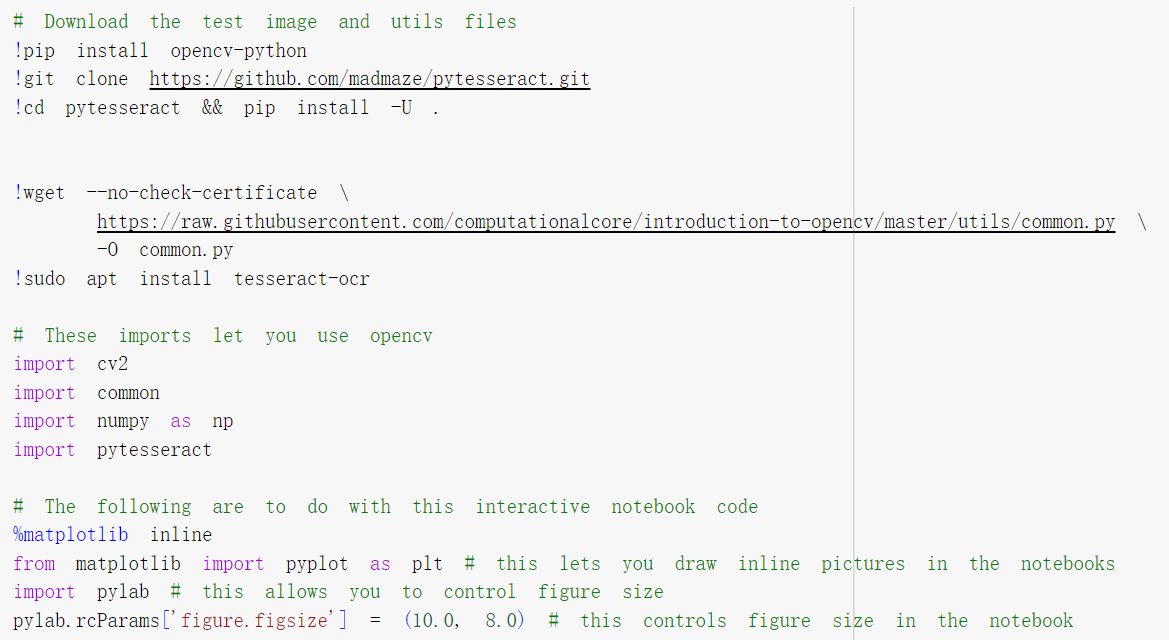
1. **Setup Environment:**

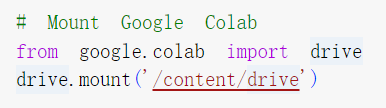
Here we used Google Colab, and we needed to install some necessary libraries for this task, including **cv2**(To use the functions of image processing and computer vision algorithms), **common**(To use some regular functions), **numpy**(For numerical calculation and processing of arrays), **pyplot**(To make sure we can show our images in Google Colab notebook), **pylab**(To control figure’s size) and **pytesseract**(To achieve text recognition).

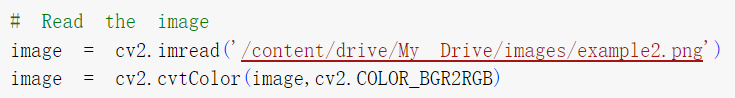


It is noticeable that in Goodle Colab, we **can not** directly use “cv2. imshow()” to display images because of its way to display that needs to pop up a window. Instead, we need to import “matplotlib” to achieve image display.

1. **Load an Image**:

Here I used an image which I captured from my Moodle. And I have already uploaded it to my Google Drive. Now we need to get the image. Before we load the image, we need to call the drive.mount() function.

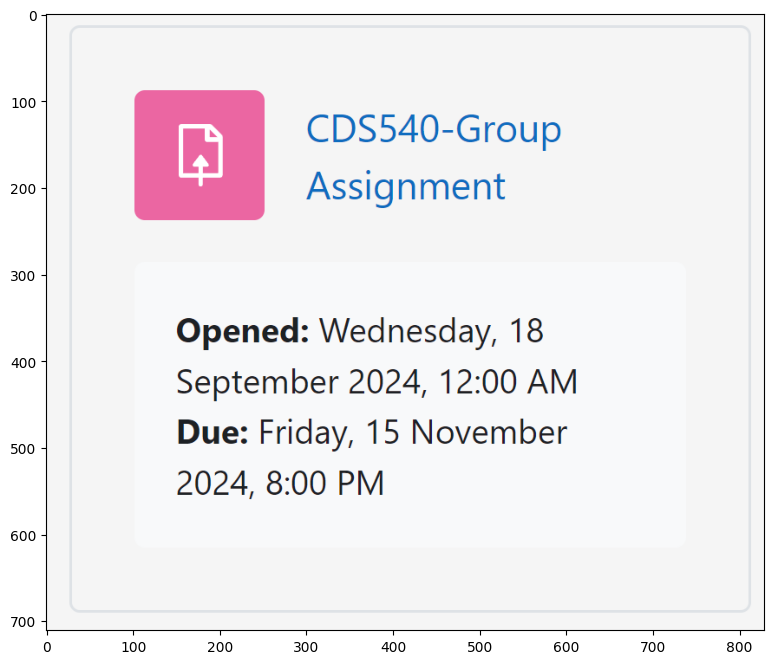




It is noticeable that cv2.imread() will read our picture in “BGR” form. We need to use cv2.COLOR\_BGR2RGB to convert it into RGB form, because the matplotlib display pictures in RGB form.

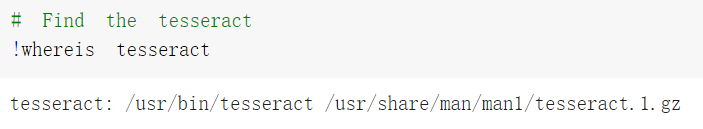
1. **Pre-process the Image**:

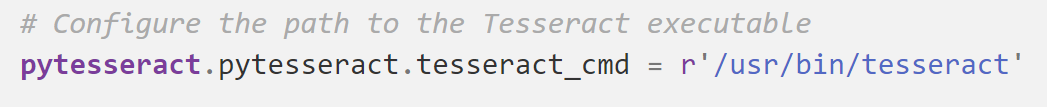
For we have picked a clear picture with clear texts, we don’t need to convert the image to grayscale or use thresholding, blurring, or edge detection to enhance text visibility. We just need to convert the image from BGR to RGB. But we can still use them to improve our text visibility effect.



1. **Text Detection**:

Here we use one of the OpenCV's text detection libraries, pytesseract, to achieve text detection.Then we also need to find the path of Tesseract-OCR because the path of the Tesseract executable must to be set.

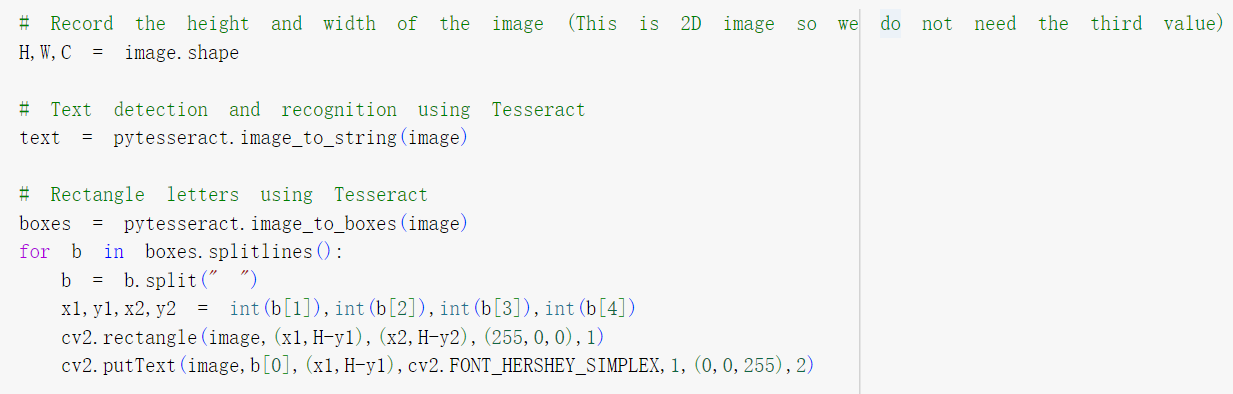




We still need to record the specific attribute of our image. We can use image.shape() to get them.



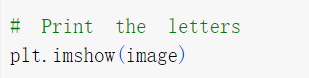
Then, we can use pytesseract.image\_to\_string(image) to get our texts. And if we need to rectangle these letters, we need to use pytesseract.image\_to\_boxes(images).

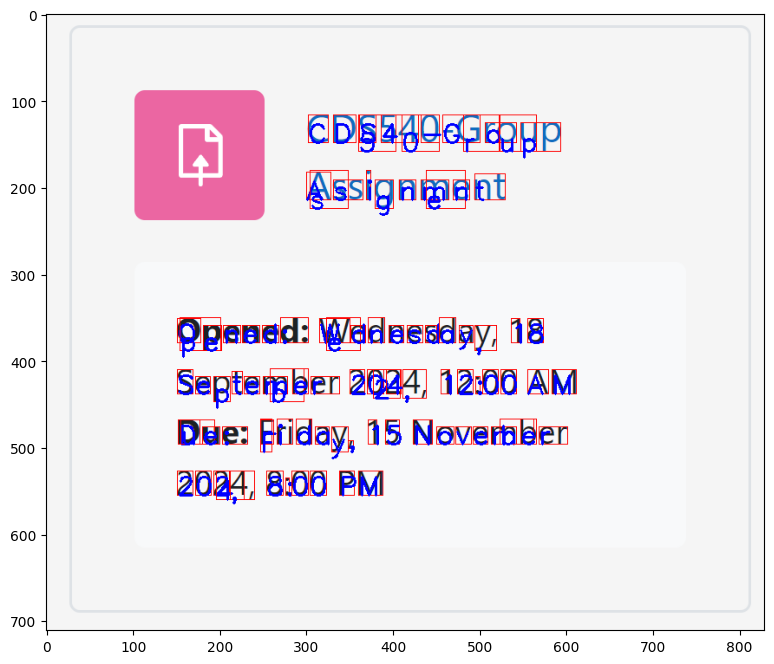


In this way, we can draw bounding boxes around detected text areas.

1. **Extract and Display Text**:

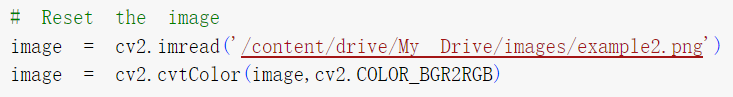
After completing the above steps, we have added the extracted text to our image. Next, we only need to use plt. imshow (image) to combine the image with the extracted text and print them out.





From the figure, we can know that we can get exact texts but sometimes the position of the texts are not totally accurate. But it is acceptable for us.

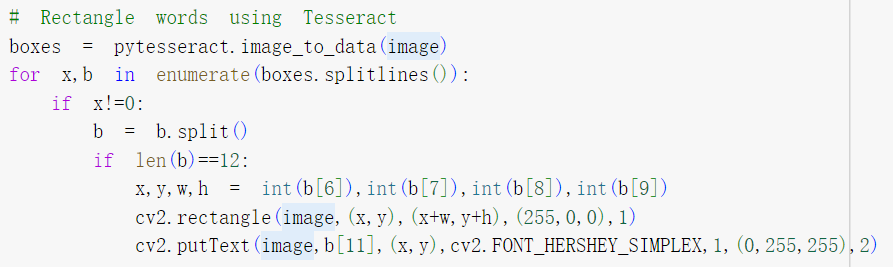
To improve the effect of our texts detection, we can rectangle these texts in word form. Actually, in another way, we can divide the data we obtained through pyteseract.image\_to\_data(image) according to its length. Firstly, we need to reset our image because the image we have right now is a new image with texts which we get in above steps.

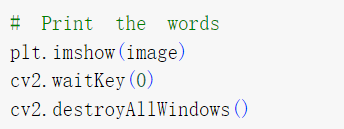


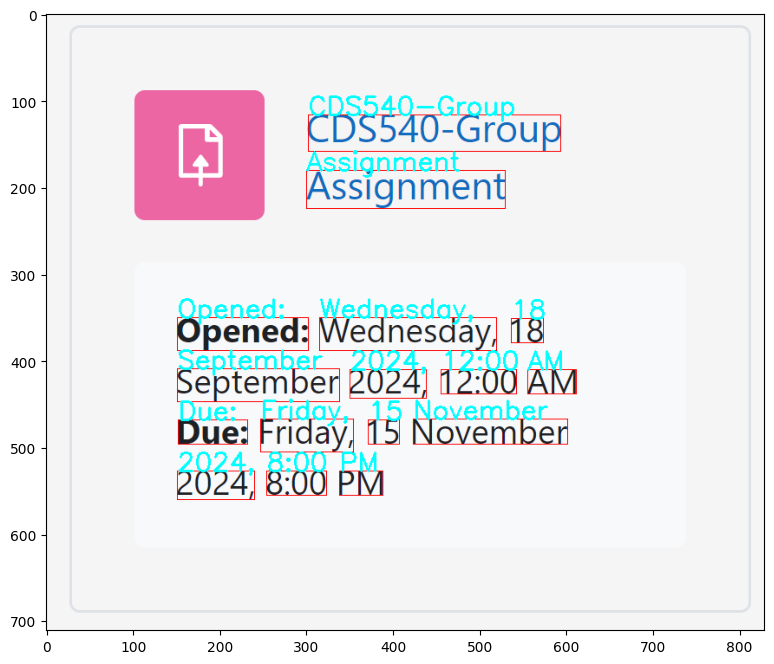
Then, we get the data of our image by using pytesseract.image\_to\_data(image).



After that, we can know that our data whose length is 12 is a integral word. So we just need to select data with a length of 12, and then box these data with rectangle to achieve word extraction.

Then we can combine the extracted words with the original image to generate a new image and print it out.



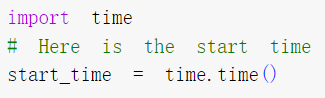


1. **Efficiency and Effectiveness Evaluation**:
2. Measure the performance of your text detection system.

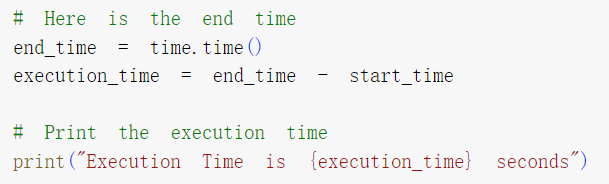
From our demonstration examples, we can see that the success rate of image recognition using this method is high. Pytesseract itself has high text detection capabilities, and the text detection system performs well.

1. Discuss the efficiency (speed) and effectiveness (accuracy) of your approach.

In order to measure the actual speed consumption of this code, we imported the time library here to accurately measure time consumption. We can add the following code to the initial position of the code.



And we can add the following code to the end of the code and calculate the code runtime.



Then, we can rerun the entire code and obtain the runtime of the code.



We can see that this scheme is fast and can complete image recognition in 25 seconds. In most cases, we still need to perform image preprocessing, such as converting to grayscale, sharpening, etc. And the image we selected is relatively clear, so extracting the text from the image is relatively simple. In practical applications, the specific time consumption may be greater than that of the proposed solution.

To test accuracy, we selected another image. This image is also relatively clear, but the density of text is higher, which may lead to a decrease in the accuracy of text extraction. Through testing, we found that a high density of text does not affect the measurement accuracy of this scheme. If the images we capture are not clear enough or too blurry, it may reduce the accuracy of text extraction. This is inevitable, as when images cannot be accurately recognized, text naturally cannot be accurately extracted.

1. Suggest potential improvements.

It is noticeable that when we are dealing with images that are not clear enough, in addition to the above steps, we still need to perform image preprocessing. For example, if we need to handle images with insufficient contrast, we need to convert the image into a grayscale image. Alternatively, if the image clarity is not high, we need to use sharpening function to improve the visibility of the image. Our method do not have these functions. But it will be easy to add them.

In addition to these preprocessing steps, it is also important that our method can not recognize Chinese characters. Considering that the complexity of Chinese characters is much higher than that of English letters, This is a significant limitation. Chinese characters are composed of multiple strokes and can vary significantly in their structure, which makes it more difficult and challenging to make our programme recognize them. Moreover, the combination of Chinese words is also more complex than that of English words. In English, words are formed by combining a relatively small set of letters in a linear fashion. In contrast, Chinese words can be formed by combining characters in various spatial arrangements, adding another layer of complexity. As a result, achieving accurate Chinese character recognition and extraction requires more comprehensive libraries and more complex algorithms.