Criteria C: Development

A/ Keys

1. "***SC***" = Success Criteria (criterion A)
2. ***Annotation Key*** (section F - K):

Colour	Type
	Processes
	Declaring variables/data structures
	Outputs

B/ General Overview

The main program is extensible as it is modular, with functions and individual files for different purposes, making it easier to modify specific features. The function and variable names are also intuitive. For example, "**make_folder"** is a function to create folders; "**path_file**" stores the path of the input file. Furthermore, as the program uses non-native libraries, a requirements.txt was made to automatically install any requirements when the program is initiated, eliminating the need for manual installation, which also fulfills **SC1**.

Type	Program Section	Program description	Complex Techniques
---	initiate_program	Run command line prompt.	- Text formatting
I/O	make_folder	Making necessary folders.	- File Handling
I/O	input_manipulation	1/ Cropping footers and merging all pages into one long image. 2/ Cropping and saving the left margin	- File Handling - Image manipulation - Data type conversion - List comprehension - PIL
find_qs	find_qs (find question and score)	Finding the coordinates of all questions and score brackets.	- OpenCV2 - Tesseract - RegEx - List comprehension
merge_qs	merge_qs	Sorting all score bracket coordinates under their corresponding question coordinate in one dictionary.	- Data structure manipulation - Nested loops and conditional statements
crop_final	crop_final	Crop each question from the longer 'merged' image based on the coordinates in the merged dictionary.	- Data structure manipulation - File handling - PIL

i/o	delete_folder	Deletes all the unnecessary folders.	- File handling
---	mark_scheme	Runs all necessary functions for parsing through a corresponding mark scheme in the same way.	- Modular programming and calling functions

C/ File Structure

Before	During	After
├── cleaning.py ├── crop_final.py ├── crop_ms.py ├── crop_table.py ├── init.py ├── input_manipulation.py ├── main.py ├── make_folder.py ├── qs_mainpulation.py ├── question_score.py ├── read_question_ms.py ├── requirements.txt ├── score.py ├── test1.pdf ├── test1_ms.pdf ├── test2.pdf ├── test2_ms.pdf ├── test3.pdf └── test3_ms.pdf	├── **MSResults** │ ├── 0417_11_M_J_Q1.jpg │ ├── ... │ └── 0417_11_M_J_Q17.jpg ├── **QPResults** │ ├── 0417_11_M_J_Q1.jpg │ ├── ... │ └── 0417_11_M_J_Q17.jpg ├── cleaning.py ├── **complete.jpg** ├── crop_final.py ├── crop_ms.py ├── crop_table.py ├── **cropped** │ ├── cropped1.jpg │ ├── ... │ └── cropped14.jpg ├── init.py ├── input_manipulation.py ├── **left** │ ├── left_tab1.jpg │ ├── ... │ └── left_tab14.jpg ├── left_long.jpg ├── main.py ├── make_folder.py ├── **og** │ ├── out0.jpg │ ├── ... │ └── out15.jpg	├── **MSResults** │ ├── 0417_11_M_J_Q1.jpg │ ├── ... │ └── 0417_11_M_J_Q17.jpg ├── **QPResults** │ ├── 0417_11_M_J_Q1.jpg │ ├── ... │ └── 0417_11_M_J_Q17.jpg ├── cleaning.py ├── crop_final.py ├── crop_ms.py ├── crop_table.py ├── init.py ├── input_manipulation.py ├── main.py ├── make_folder.py ├── qs_mainpulation.py ├── question_score.py ├── read_question_ms.py ├── requirements.txt ├── score.py ├── test1.pdf ├── test1_ms.pdf ├── test2.pdf ├── test2_ms.pdf ├── test3.pdf └── test3_ms.pdf

	├── qs_mainpulation.py ├── question_score.py ├── read_question_ms.py ├── requirements.txt ├── score.py ├── **table** │ ├── header.jpg │ ├── table1.jpg │ ├── ... │ └── table21.jpg ├── test1.pdf ├── test1_ms.pdf ├── test2.pdf ├── test2_ms.pdf ├── test3.pdf └── test3_ms.pdf	

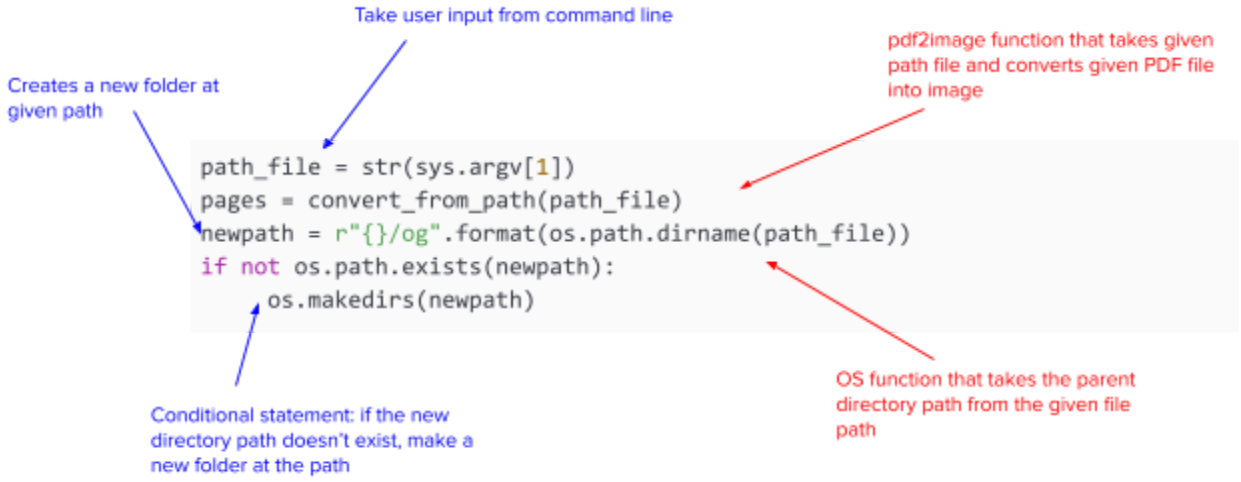
D/ Python Libraries Used

Library Name	Version	Function
Progress	1.6	CLI: Shows a progress bar for processing period
Rich	10.12.0	CLI: Allows rich text formatting for CLI in order to make it more user friendly and to segment different sections of the CLI
cv2	4.5.1.48	Supports computer vision, but will be used to manipulate image colour thresholds and filtering in order to support PyTesseract's OCR function.
pdf2image	1.14.0	Library used to convert pdf documents into PIL images. The "convert_from_path" function will be used.
pytesseract	0.3.7	Optical Character Recognition (OCR) library for python; a more heavily used library that reads and picks up on text and its corresponding data from PIL images.
PIL	8.1.1	Python's imaging library; the Image module is heavily used for its ability to process and "open" images.

numpy	1.20.1	Used once in tandem with cv2 to apply a sharpen filter.
re		Used once to filter data from tesseract.
os		Used to check the existence of paths and make new directories.
sys		Used to grab command line arguments.
shutil		Used to remove non-empty directories.

E/ Handling folders

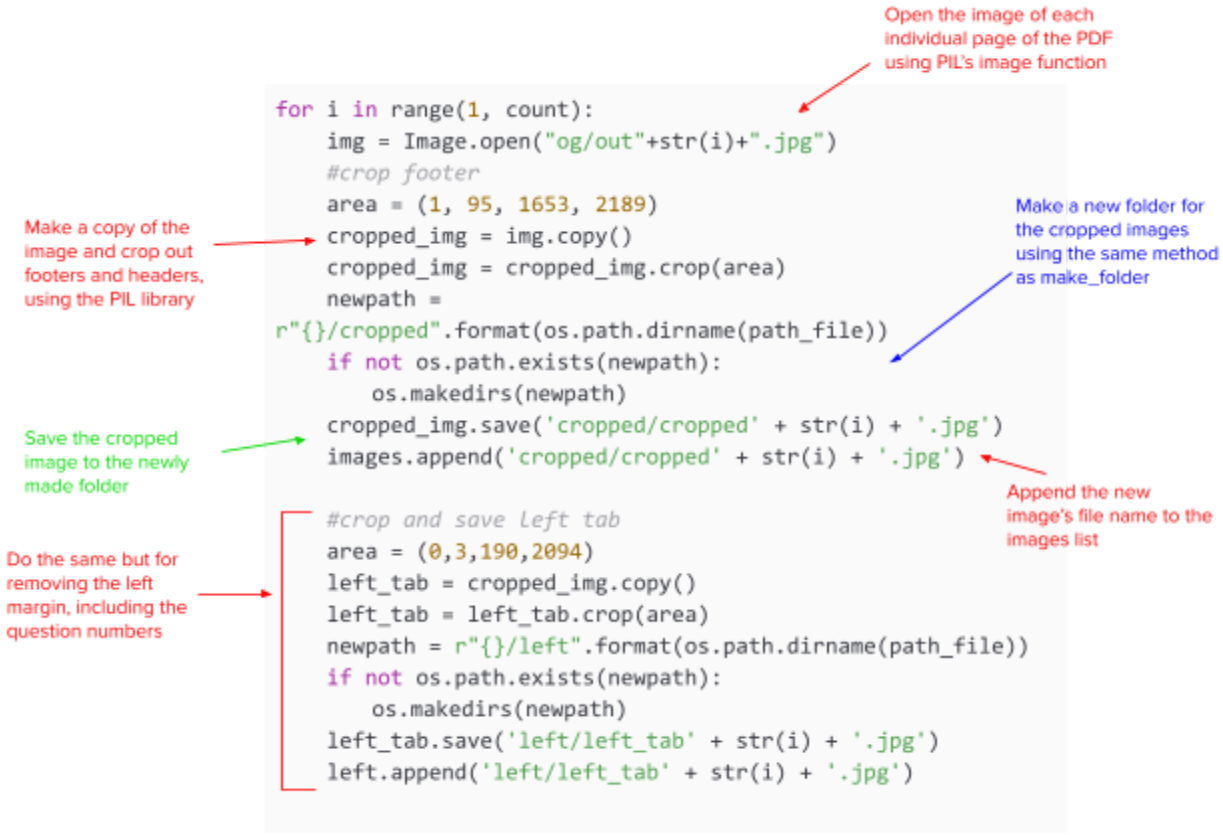
As per **SC7**, at each step of the process where new images are being made, a new folder is created (**make_folder**) in order to keep all the files organised.



The **make_folder** function first converts the PDF (user input) into images using **pdf2image**'s **convert_from_path** function, which are stored in a new directory. This is done through the **os** library to find the parent directory path from the user input. The function also uses a **conditional statement** in the case that the directory has not been made already. This will be useful for the **mark_scheme** function later in the code and to store images in folders with names indicating their progress.

F/ Image manipulation

i/ cropping the footer




```
for i in range(1, count):
    img = Image.open("og/out"+str(i)+".jpg")
    #crop footer
    area = (1, 95, 1653, 2189)
    cropped_img = img.copy()
    cropped_img = cropped_img.crop(area)
    newpath =
r"{}/cropped".format(os.path.dirname(path_file))
    if not os.path.exists(newpath):
        os.makedirs(newpath)
    cropped_img.save('cropped/cropped' + str(i) + '.jpg')
    images.append('cropped/cropped' + str(i) + '.jpg')

    #crop and save left tab
    area = (0,3,190,2094)
    left_tab = cropped_img.copy()
    left_tab = left_tab.crop(area)
    newpath = r"{}/left".format(os.path.dirname(path_file))
    if not os.path.exists(newpath):
        os.makedirs(newpath)
    left_tab.save('left/left_tab' + str(i) + '.jpg')
    left.append('left/left_tab' + str(i) + '.jpg')
```

As the footer disrupts the merging process, it has to be cropped from each page. Though this isn't a required **SC**, removing the footer makes subsequent processes more straightforward.

To crop the identical footers, a defined area is used and new copies of the image are made. The images are converted into a **PIL image object** as the dimensions are needed for later. They are then saved in a new folder that will be used in **merging**.

The same is then done for the left margin.

ii/ merging images

As the structure of the question paper (Crit. B) is such that the left margin is only consistent with the beginning of each “main” question and not its sub-questions, it becomes difficult to crop out each separate sub-question. Therefore, in order to fulfill **SC4**, all cropped images are stitched together to make a vertically concatenated image.

List comprehension to open each image in the list as a PIL image object

```
#merge all cropped images into one long jpg for easier cropping
imgs = [Image.open(i) for i in images]
min_img_width = min(i.width for i in imgs)
total_height = 0

for i in range(len(images)):
    total_height += imgs[i].height
```

Find the minimum width from the list of PIL image objects

Find total height for the long image by summing all of the heights from the list of PIL image objects

For merging, each image is opened as a PIL object, so the data can be read from the file once it is called upon (**i.width**) (*Image Module*). A total height and minimum width is calculated from the dimensions of the image objects, which are then used to define a new blank image for the pages to be pasted on, creating a long, merged image.

```
long = Image.new(imgs[0].mode, (min_img_width, total_height))
y=0
for img in imgs:
      long.paste(img, (0, y))
      y+= img.height
      if y>32767:
       long.thumbnail([32767, 32767], Image.ANTIALIAS)
long.save('long.jpg')
```

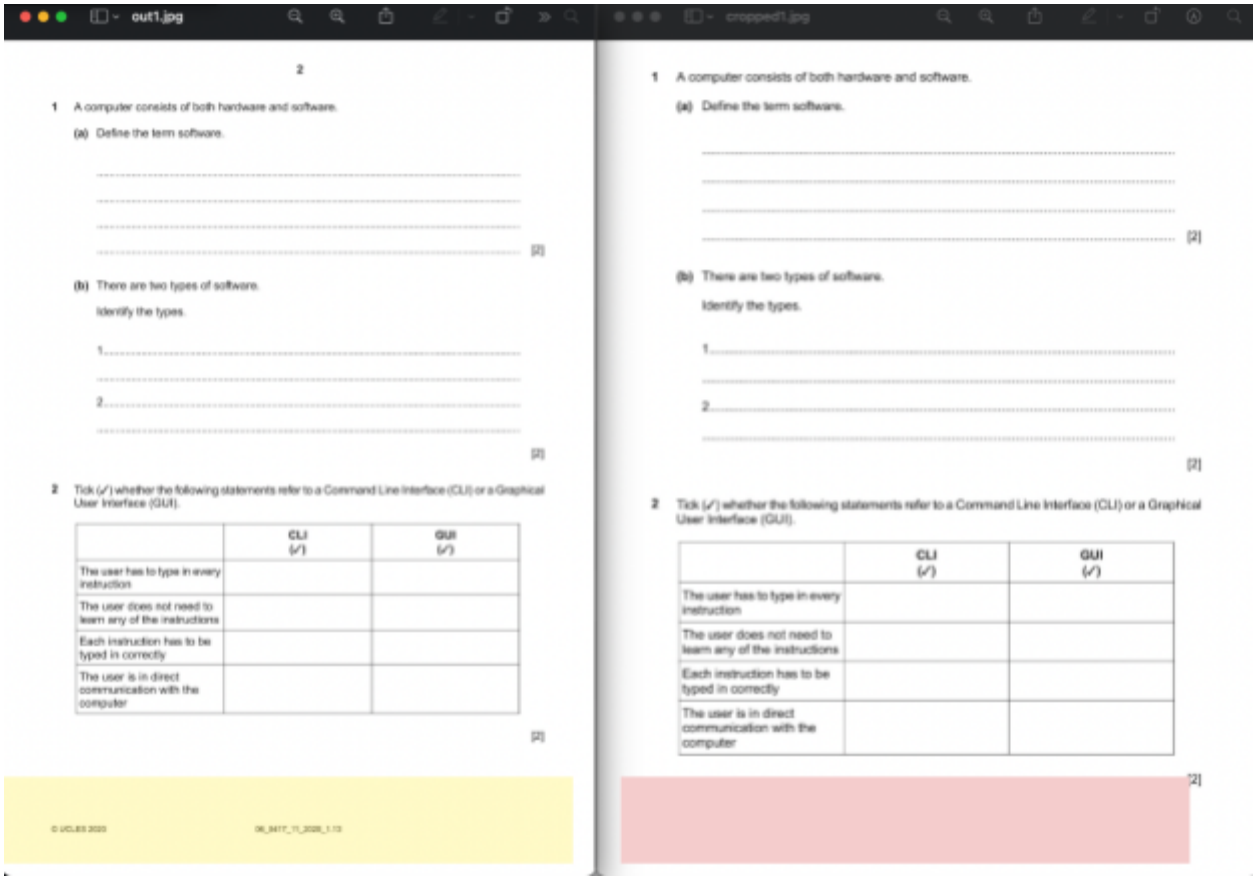
Create a new blank image using PIL function with the minimum image width and total height

Paste each individual image on sequentially onto the new blank image

If the total height of the merged images is greater than the boundaries of the image, resize using thumbnail function

Save the final merged image as a new image

If the total height exceeds the maximum of 32767, the image needs to be resized through the **PIL thumbnail** function, which retains the original aspect ratio. The final image is saved as a new image named “long.jpg”.



Difference between original (left) and cropped page (right)

```
Image:
  Filename: long.jpg
  Format: JPEG (Joint Photographic Experts Group JFIF format)
  Mime type: image/jpeg
  Class: DirectClass
  Geometry: 1652x31410+0+0
  Units: Undefined
  Colorspace: sRGB
  Type: TrueColor
  Base type: Undefined
  Endianness: Undefined
  Depth: 8-bit
  Channel depth:
    Red: 8-bit
    Green: 8-bit
    Blue: 8-bit
```

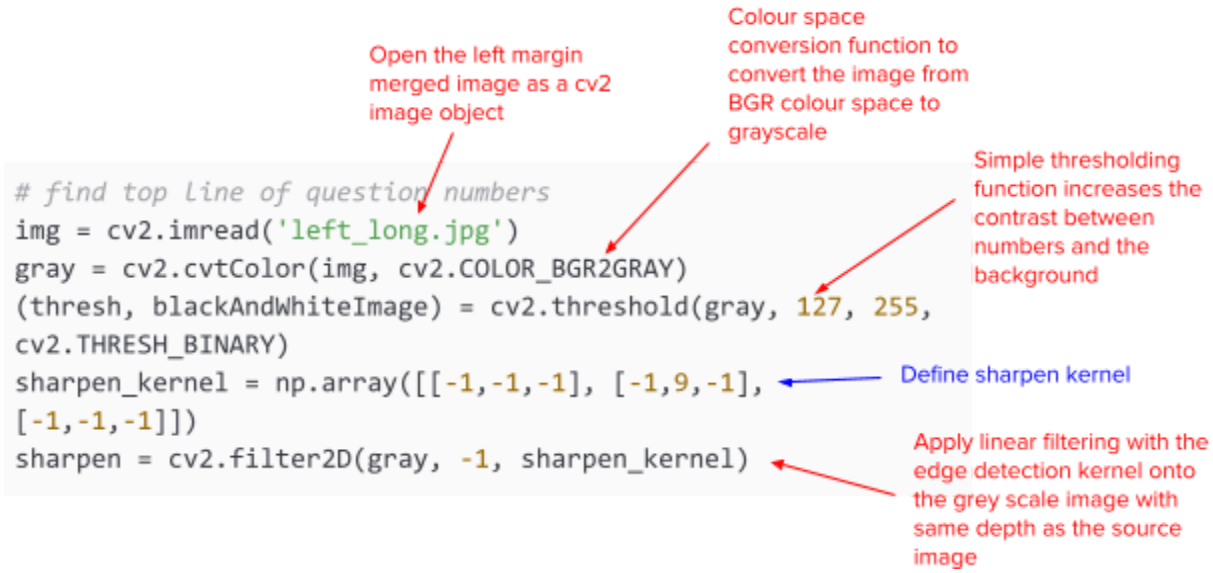
*Information of merged image, shown using **ImageMagick** library's **identify (-verbose)** function*

G/ OCR

i/ Find questions:

Preprocessing:

As **tesseract** is open source, the output is sometimes inaccurate. To minimise this, the images are preprocessed through converting colour space, sharpening, dilating and eroding. *('Improving the Quality of the Output')*




```
# find top line of question numbers
img = cv2.imread('left_long.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
(thresh, blackAndWhiteImage) = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY)
sharpen_kernel = np.array([[-1,-1,-1], [-1,9,-1],
[-1,-1,-1]])
sharpen = cv2.filter2D(gray, -1, sharpen_kernel)
```

The process is as follows:

1. Convert image to grayscale

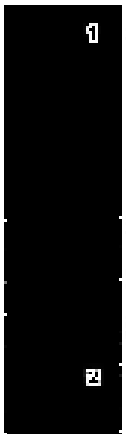
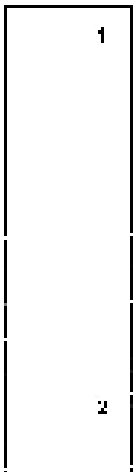
The image is read as a **cv2 image object** then converted from BGR to grayscale using cv2's **cvtColor** with the flag **COLOR_BGR2GRAY** to increase the contrast between words and whitespace (*Color Conversions — OpenCV Documentation*). The resulting grayscale image is stored as **gray**.

2. Apply thresholding function to reduce 'fuzz'

For the simple thresholding function, the arguments are the source image (**gray**), the **threshold value**, the **maxVal**, and an **OpenCV** provided thresholding style, which creates a high-contrast image (*Image Thresholding — OpenCV-Python Tutorials Beta Documentation*). For the flag **cv2.THRESH_BINARY**, objects appear as black on a white background *('OpenCV Thresholding (Cv2.Threshold)')*. The function converts any pixel with a value greater than the second argument (ie. 127) into the third argument (255). In doing so, any pixel that is more gray than black/white will be converted into white, removing 'fuzzy' edges.

3. Apply sharpening kernel to sharpen the image

A **sharpening kernel**, which emphasises the shape of lines, is then applied to the **grayscale** image using linear filtering. A kernel is a matrix that converts the pixels of the image (*Carvalho*) around an 'anchor' point, which in this case is in the center at 9. The most common generic sharpening kernel (anchor@8) was not used because it failed to sharpen the image enough for all numbers to be read, as shown below:

Kernel used	Image produced
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

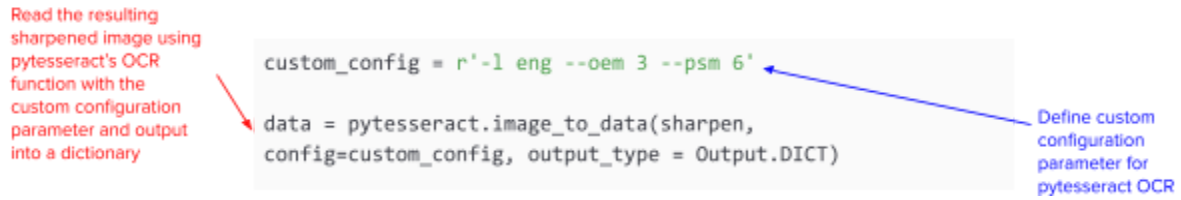
Essentially, the kernel takes 3x3 pixels and multiplies it by the corresponding number in the matrix. The sum of the products then represents the converted image's anchor pixel *('Image Kernels Explained Visually')*.

Linear filtering (**filter2D()**) is then used to apply the kernel. It's a function specific to **OpenCV** which requires a source image, the depth of the resulting image, and the kernel. When depth is

assigned -1, the resulting image should have the same bit-depth as the source image (*Image Filtering Using Convolution in OpenCV | LearnOpenCV #*).

OCR (pytesseract):

OCR can be applied now that the image has been preprocessed. The OCR engine chosen was **pytesseract** as it easily outputs data, unlike Google Vision, and the text needed to be collected was of a consistent font, format and are typically numbers.



The custom configuration used was `r'-l eng --oem 3 --psm 6'`, which can be broken down into 3 parts:

`-l eng`	Language flag. Specifies what language the engine should be expecting/reading.

`--oem 3`	oem = OCR Engine Mode
	Tesseract has multiple OEMs with different uses and speeds. As default, 3 is used.
	
	Options:
	`OCR Engine modes:`
	`0 Legacy engine only.`
	`1 Neural nets LSTM engine only.`
	`2 Legacy + LSTM engines.`
	`3 Default, based on what is available.`

`--psm 6`	psm = Page Segmentation Mode
	- specifies how tesseract splits and reads the source image in terms of lines of text and words.
	- defaults to psm 3 if not specified.
	
	PSM6 was chosen as the source image was a long image of sparse numbers.
	
	Options:
	`Page segmentation modes:`
	`0 Orientation and script detection (OSD) only.`
	`1 Automatic page segmentation with OSD.`
	`2 Automatic page segmentation, but no OSD, or OCR. (not implemented)`
	`3 Fully automatic page segmentation, but no OSD. (Default)`
	`4 Assume a single column of text of variable sizes.`
	`5 Assume a single uniform block of vertically aligned text.`
	`6 Assume a single uniform block of text.`
	`7 Treat the image as a single text line.`
	`8 Treat the image as a single word.`
	`9 Treat the image as a single word in a circle.`
	`10 Treat the image as a single character.`
	`11 Sparse text. Find as much text as possible in no particular order.`
	`12 Sparse text with OSD.`
	`13 Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.`

For loop to clean up the resulting image data dictionary

Convert dictionary into list if the text variable is a digit (page number)

Failsafe for if pytesseract reads the number 1 as 4 or 7

```
for i in list(data):
    if i not in ["text", "height", "width", "top", "left"]:
        data.pop(i, None)
question_result = [(left, top, width, height, text) for left,
top, width, height, text in zip(*data.values()) if
text.isdigit()]
if list(question_result[0])[4] != '1':
    new_tuple = list(question_result[0])
    new_tuple[4] = '1'
    new_tuple = tuple(new_tuple)
    question_result[0] = new_tuple
```

The data output from **pytesseract.image_to_data** is extremely thorough (*Lee*) and needs cleaning up, done with a **for loop**, to only include: the text itself (**SC3d)**, height, width, top, and left. The four pixel values indicate the coordinates of the text, thus giving us the information needed for **cropping**.

When the bulk has been removed, the remaining **dictionary** is filtered for page numbers (numerical text). However, after much testing, it seems that **tesseract** reads “1” as 7. To solve this, a failsafe was implemented so that the first numerical text is always a ‘1’.

ii/ Find scores:

To locate all of the score brackets to find where each main question ends, **pytesseract** is used again.

Open the left margin merged image as a PIL image object

Define custom configuration parameter for pytesseract OCR

Read the resulting sharpened image using pytesseract's OCR function with the custom configuration parameter and output into a dictionary

For loop to clean up the resulting image data dictionary

Compile RegEx pattern into object

Convert dictionary into list if the text variable matches RegEx pattern

```
#find bottom line of score numbers
img = Image.open("long.jpg")

custom_config = r'-c tessedit_char_whitelist=[]1234567890 --psm 11'
data = pytesseract.image_to_data(img, config=custom_config,
output_type = Output.DICT)
for i in list(data):
    if i not in ["text", "height", "width", "top", "left"]:
        data.pop(i, None)
pattern = re.compile("\[(.*?)\]")
score_result = [(left, top, width, height, text) for left, top,
width, height, text in list(zip(*data.values())) if
pattern.match(text)]
```

The custom configuration used was `r'-c tessedit_char_whitelist=[]1234567890 --psm 11'`, which can be broken down into 2 parts:

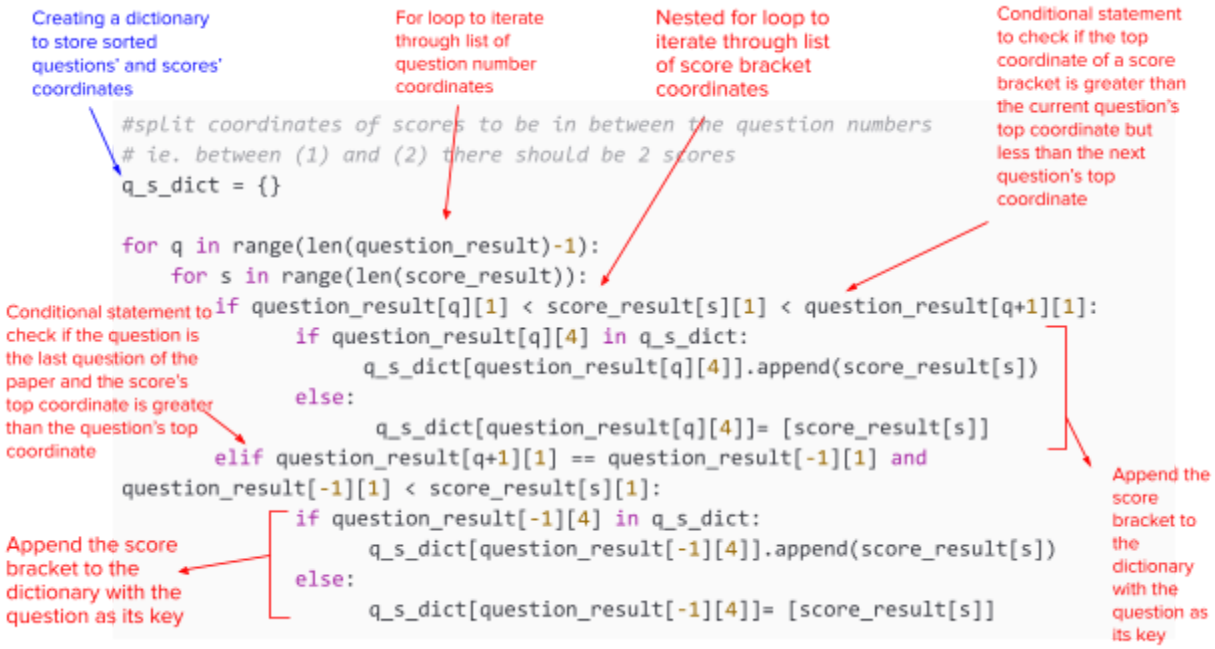
`-c tessedit_char_whitelist=[]1234567890`	char_whitelist = limits detection - []123456789 limits detection to only score brackets as they are consistently "[num]"
`--psm 11`	psm = Page Segmentation Mode PSM11 was chosen as it was the most accurate after rounds of testing. **options same as above*

Regular Expression (RegEx) is a library used to match instances of text. It compiles a **'pattern'** to compare against inputs and returns a match object depending on if the text matches using **pattern.match()**. If no text matches, it will return **None**, which allows us to use a **conditional statement** (*Re — Regular Expression Operations — Python 3.10.0 Documentation*). The expression used was: **"`\[([0-9]*?)\]`"** which can be interpreted as:

`\[`	Match "[" character
`()`	Captures any character besides line terminator in a group
`[0-9]`	Matches any digit
`*?`	Match any instances of the previous term ([0-9]) between zero and unlimited times until the next first instance of "]"
`\]`	Match "]" character

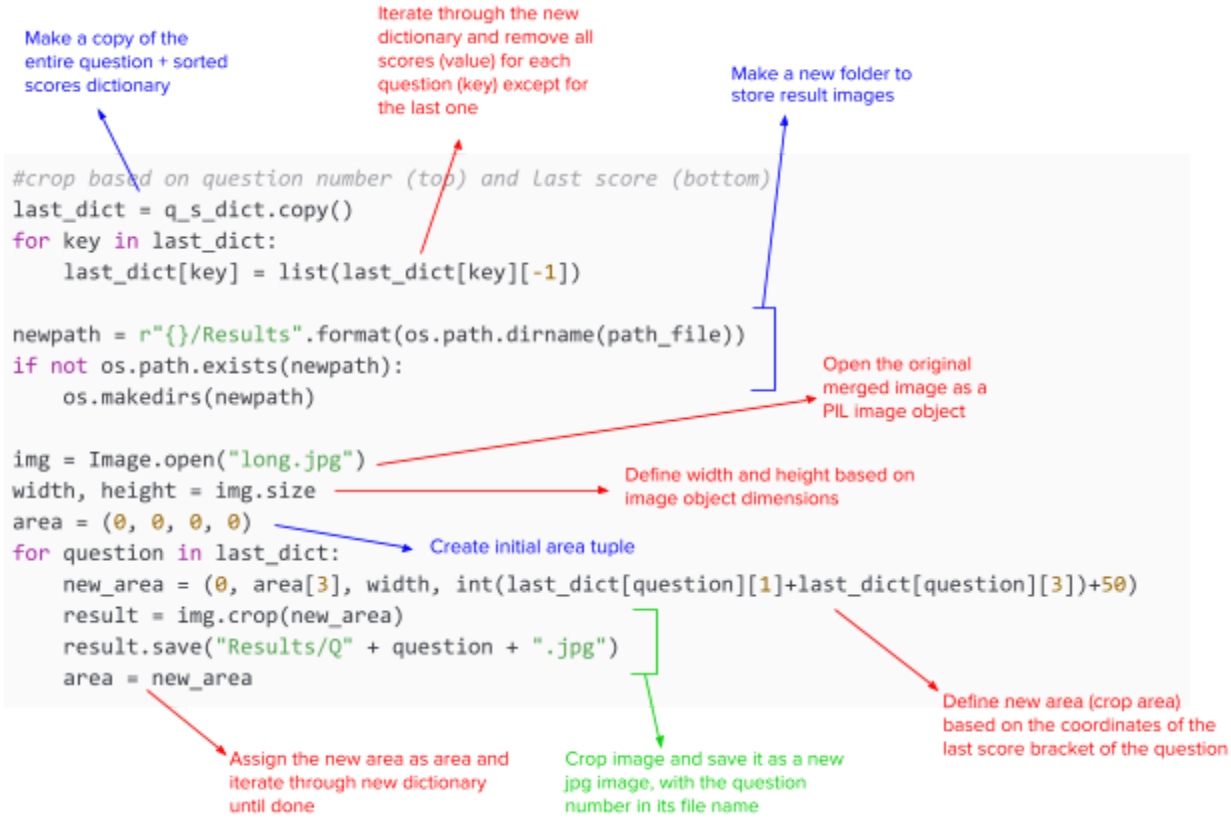
H/ Data manipulation

This fulfills **SC4** as each sub-question is sorted under their corresponding main question. **Nested for loops** and **conditional statements** are used.



All results are stored in a **dictionary** with question numbers as keys. Sub-question score brackets are then sorted under the correct key by comparing their (x, y) coordinates with the main question coordinates. All questions are then in one organised data structure for easier further data manipulation.

I/ Output handling



The **dictionary** containing all sorted sub-questions and coordinates can be then simplified using **list comprehension** to only contain the last sub-question for each main question.

The merged image is opened as a **PIL object** and cropped using a **for loop** to loop through the **dictionary** with the required coordinates. The image is cropped based on the top coordinates of each question and the bottom coordinates of its last sub-question. Using a **dictionary** helps in storing the coordinates logically and naming the files using the question number **keys**.

Once all result images are saved, **SC3, 4, and 6** will be fulfilled.

Word Count: 1157

J/ Bibliography

Carvalho, Thiago. 'Basics of Kernels and Convolutions with OpenCV'. *Medium*, 10 July 2020, https://towardsdatascience.com/basics-of-kernels-and-convolutions-with-opencv-c15311ab8f55.

Color Conversions — OpenCV Documentation. https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/page_imgproc_color_conversions.html#doxid-de-d25-imgproc-color-conversions-1color-convert-rgb-gray. Accessed 12 Nov. 2021.

Image Filtering Using Convolution in OpenCV | LearnOpenCV #. 8 June 2021, https://learnopencv.com/image-filtering-using-convolution-in-opencv/.

'Image Kernels Explained Visually'. *Explained Visually*, https://setosa.io/ev/image-kernels/. Accessed 12 Nov. 2021.

Image Module. https://pillow.readthedocs.io/en/stable/reference/Image.html. Accessed 12 Oct. 2021.

Image Thresholding — OpenCV-Python Tutorials Beta Documentation. https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html. Accessed 12 Oct. 2021.

'Improving the Quality of the Output'. *Tessdoc*, https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html. Accessed 12 Nov. 2021.

Lee, Matthias. *Pytesseract: Python-Tesseract Is a Python Wrapper for Google's Tesseract-OCR*. 0.3.8. *PyPI*, https://github.com/madmaze/pytesseract. Accessed 12 Nov. 2021.

'OpenCV Thresholding (Cv2.Threshold)'. *PyImageSearch*, 28 Apr. 2021, https://www.pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/.

Re — Regular Expression Operations — Python 3.10.0 Documentation. https://docs.python.org/3/library/re.html#re.Pattern.match. Accessed 12 Nov. 2021.

Shutil — High-Level File Operations — Python 3.10.0 Documentation.

https://docs.python.org/3/library/shutil.html. Accessed 12 Oct. 2021.