Interview 1

Date: January 27th, 2021
S - Student
C - Client

S: What’s the problem?

C: So for IGCSE ICT, I've been working on having unit based questionbanks that I can share with the students so that they can practice ready for the final exams at the end of the course. So far, I've got a bunch of multiple choice questions that I've come up with that are organised by unit. What I don't have is the step beyond multiple choice questions where they are doing actual exam questions. So what I really want is to be able to pull questions from past papers that I can categorise by unit so that I could pull out, let’s say 5 security questions, drop them into a document, push them to the kids, and they can work on it. That's really what I'm after.

S: How have you been doing this before?

C: Manually. So the snipping tool or exporting PDFs as a PNG and literally cropping them. Which is a lot.

S: So if I made a shell script that you could just put all the PDFs into a folder and run it on and you get a bunch of sorted out questions…

C: That would be perfect. The way that it works is each paper has a unique name so the file name for each paper is unique. If at the end of that we could append dash Q1, Q2, Q3… all the way through, that would be brilliant. The only thing that complicates that a little bit is that there are matching mark schemes. So separate documents and very different page ordering, obviously there's more information on a mark scheme than there is on a question paper. So the absolute ideal would be that I could pick out question paper question 1 and the mark scheme question 1 and the two of those images cover the whole question.

That’s going to be complicated by page breaks. Because a lot of the time you'll have a question on one page, and they try really hard to make sure they don’t put questions on more than one page, but they'll very often have the mark scheme going over multiple pages. So that might be really tricky, I’m not sure…like if we had question 1 dash 1, 1 dash 2 for page 1, page 2 that could work.

S: Would you want the sub questions like A and B to be separated or as one full question?

C: I very rarely separate out sub-questions if I'm going to give them a question. IGCSE tends not to have, like ITGS does, part A’s about one thing and part B’s about a totally different thing. They don't do that in IGCSE. So like question 6, even if it's got four parts, all of those four parts will be about the same thing. So I wanna keep them probably as one part. But if it were easier to have 1A, 1B, 1C, I would be fine with that too. It wouldn't make it any harder for me to put them together, so whatever works for you.

S: Just for the IGCSE papers then?

C: Just for the IGCSE papers.

The other thing though is IGCSE is done by Cambridge and I'll be teaching Cambridge A levels. So if it works for the IGCSE papers, they're going to have a consistent format, so it would work for A levels as well. So that'll be awesome.

S: Do you need a watermark on the paper with like what paper it's from?

C: Oh, that might be helpful.

S: I don't know if I've asked this already but do you need it with a GUI or would you be fine with just a script?

C: It would be fine with just a shell script. The ideal would be not having to customise per paper. The only reason I can think of to have a GUI is if there were some different options and stuff. The papers are pretty consistent, or at least they have been, since the syllabus change in 2016. They used to be a little different before that but now they're all super super consistent, which is good. I'm not really interested in also doing the papers pre-2016 so I don't really think adding options for it would be helpful. Like would it benefit you to put a GUI on it? I don't see any benefit to me in having a GUI. I mean, literally it would be like a drop the files here, hit go. I don't know what else could be put into the process that could make it better.

S: Do you want it sorted by year or by topic?

C: Well, if the file name is basically going to be appended to the current file name of the existing files, literally the only thing you would have to contribute at all is the appended bit, so like Q1.

S: So it's okay if I set up 5 folders and each one would have the questions for each year?

C: Yep, I mean you could. You don't necessarily have to put them in folders because each paper is uniquely identified. So the way that works is, let me show you… '0417' is the course code, '_s20' means Summer 2020, 'ms' is mark scheme, 'qp' is question paper and paper 11, 12, 13, 21, 22, 23, 31, 32, 33, like each of these are uniquely identified. So if all you were doing in terms of file structure was appending it to this '- q1' then I could drop this whole folder in there and they would still be uniquely identified as that exam paper. If you wanted to put a folder structure that would be fine too but it wouldn't be necessary for me to know where it came from.

Interview 2

Date: January 9th, 2022

Note: Due to my client relocating, this conversation was had over email with him being shown my Criterion D video

C - Client

C: The UI is easy to use for the current client, but if he wants to share it with other users the setup (installing Python and such) may need some streamlining. This could mean just having a better instructional guide on how to install/use, or creating a packaging tool and more traditional user interface (GUI). It might be nice later on to add a tool which can detect point numbers of the questions and help create tests given a specific number of points. All criteria have been met, so feel free to make the appropriate remarks to that effect.

Overall Program

main.py

```
import os.path, sys, signal, re, os
from concurrent.futures import ThreadPoolExecutor
from threading import Event
from rich import print
from rich.console import Console
from rich import box
from rich.table import Table
import numpy as np
import init, make_folder, input_manipulation, question_score,
qs_mainpulation, crop_final, cleaning, crop_table, read_question_ms,
crop_ms

from rich.progress import (
      BarColumn,
      Progress,
      TaskID,
      TextColumn,
      TimeElapsedColumn
)

progress = Progress(
      TextColumn("[bold blue]Past Paper(s)", justify="right"),
      BarColumn(bar_width=None),
      "[progress.percentage]{task.percentage:>3.1f}%",
      "•",
      TimeElapsedColumn(),
)

done_event = Event() #threading

def handle_sigint(signum, frame):
      done_event.set() #terminate thread

signal.signal(signal.SIGINT, handle_sigint)
console = Console()

def run_pdf(task_id: TaskID, qp_input_path: str, ms_input_path: str,
output_format: str) -> None:
      if ms_input_path != None:
      progress.console.log(f"Processing {qp_input_path} and
{ms_input_path}")
      else:
      progress.console.log(f"Processing {qp_input_path}")
      progress.update(task_id, total = 41)
      progress.start_task(task_id)

      (images, count) = make_folder.io_mac(qp_input_path)
      progress.update(task_id, advance=2)
      (crop_bottom, count) = input_manipulation.tessa(images, count)
      progress.update(task_id, advance=28)
      (cropped, left) = input_manipulation.cropping(images, count,
crop_bottom)
      progress.update(task_id, advance=2)
      input_manipulation.merge(cropped, left)
      progress.update(task_id, advance=1)
      (question_result, score_result) = question_score.find_q_s()
      progress.update(task_id, advance=3)
      q_s_dict = qs_mainpulation.sort_qs(question_result, score_result)
      progress.update(task_id, advance=1)
      paper_code = crop_final.output_crop(q_s_dict, output_format)
      progress.update(task_id, advance=2)

      if ms_input_path == None:
      cleaning.clean_up("QP")
      progress.update(task_id, advance=2)

      elif ms_input_path != None:
      progress.update(task_id, total = 101)
      cleaning.clean_up("QP")
      progress.update(task_id, advance=1)
      (images, count) = make_folder.io_mac(ms_input_path)
      progress.update(task_id, advance=1)
      ms_question_dict = {}
      count_temp = 1
      ms_start = crop_table.check_GMP(images)
      progress.update(task_id, advance=5)
      for ms_count in range(ms_start, len(images)):
            boxes = crop_table.find_table(ms_count)
            (count_temp, box_dict) = crop_table.sort_table(count_temp,
boxes)
            ms_question_dict.update(box_dict)
            images = [images[index] for index in range(count)]
            width = crop_table.cropping(ms_count, box_dict,
np.array(images))
      progress.update(task_id, advance=50)
      (x, y, x2, y2) = crop_table.save_header(box_dict, images)
      question_dict = {}
      progress.update(task_id, advance=1)
      for ms_2_count in range(1,count_temp):
            question_result = read_question_ms.question_OCR(width,
ms_2_count)
            question_dict = read_question_ms.sort_questions(question_dict,
question_result)
      progress.update(task_id, advance=1)
      (question_dict, pages_depth) = crop_ms.sorting_ms_dict(question_dict)
      progress.update(task_id, advance=1)
      crop_ms.cropping_ms(paper_code, x, y, x2, y2, question_dict,
pages_depth, output_format)
      progress.update(task_id, advance=1)
      cleaning.clean_up("MS")
      progress.update(task_id, advance=1)

      if done_event.is_set():
      return

def start(input_file_structure: str, qp_input_path: str, ms_input_path:
str, output_format: str):
      if output_format == "1":
      output_format = "jpg"
      elif output_format == "2":
      output_format = "png"
      with progress:
      with ThreadPoolExecutor(max_workers=1) as pool:
            if ms != None:
                  task_id = progress.add_task("Parsing question paper
({qp_input_path}) and markscheme ({ms_input_path})", qp_input_path = qp,
ms_input_path = ms, start = False)
                  pool.submit(run_pdf, task_id, qp, ms, output_format)
            else:
                  task_id = progress.add_task("Parsing question paper
({qp_input_path})", qp_input_path = qp, ms_input_path = ms, start = False)
                  pool.submit(run_pdf, task_id, qp, ms, output_format)


if __name__ == "__main__":
      os_req = init.start()
      os.system("clear")
      if os_req == True:
      #input file structure
      console.rule("[bold blue]Input File Structure selection", style="bold 
white")
      console.print("Please choose [bold magenta]one[/] of the below Input 
File Structure formats for the program to run on:\n")

      table = Table(title="Input File Structure Choices", 
box=box.HEAVY_EDGE)
      table.add_column("Option", justify="center")
      table.add_column("Output format", justify="center")
      table.add_row("1", "Single Question Paper file")
      table.add_row("2", "Pair of Question Paper and Mark Scheme")
      console.print(table)

      input_file_structure = console.input("Enter [bold magenta]Option[/] 
here: ")
      while input_file_structure != "1" and input_file_structure != "2":
            input_file_structure = console.input("Usage:\n\tEnter [bold 
magenta]1 or 2[/] here: ")

      console.clear()

      #input selection
      console.rule("[bold blue]Input selection", style="bold white")
      if input_file_structure == "1":
            qp = console.input("Please type the [bold magenta]file 
name[/bold magenta], including [bold magenta]file extension (ie. .pdf)[/] 
of the past question paper file you would like to parse: ")
            while not os.path.isfile(qp):
                  qp = console.input("Usage:\n\tPlease type a [bold 
magenta]valid file name[/bold magenta], including [bold magenta]file 
extension (ie. .pdf)[/] of the past question paper file you would like to 
parse: ")
            ms = None
      else:
            qp = console.input("Please type the [bold magenta]file 
name[/bold magenta], including [bold magenta]file extension (ie. .pdf)[/], 
of the [bold magenta]past question paper file[/] you would like to parse: 
")
            while not os.path.isfile(qp):
                  qp = console.input("Usage:\n\tPlease type a valid [bold
magenta]file name[/bold magenta], including [bold magenta]file extension
(ie. .pdf)[/], of the [bold magenta]past question paper file[/] you would
like to parse: ")

            console.print()

            ms = console.input("Please type the [bold magenta]file
name[/bold magenta], including [bold magenta]file extension (ie. .pdf)[/]
of the [bold cyan]matching markscheme file[/] you would like to parse: ")
            while not os.path.isfile(ms):
                  ms = console.input("Usage:\n\tPlease type a valid [bold
magenta]file name[/bold magenta], including [bold magenta]file extension
(ie. .pdf)[/] of the [bold cyan]matching markscheme file[/] you would like
to parse: ")

      console.clear()

      #output format
      console.rule("[bold blue]Output Format selection", style="bold
white")
      console.print("Please choose [bold magenta]one[/] of the below output
formats:\n")

      table = Table(title="Output Format Choices", box=box.HEAVY_EDGE)
      table.add_column("Option", justify="center")
      table.add_column("Output format", justify="center")
      table.add_row("1", "jpg")
      table.add_row("2", "png")
      console.print(table)

      output_format = console.input("Enter [bold magenta]Option[/] here: ")
      while output_format != "1" and output_format != "2":
            output_format = console.input("Usage:\n\tEnter [bold magenta]1
or 2[/] here: ")

      console.clear()

      if input_file_structure == "1":
            console.rule("\n[bold blue]Great! Please wait while we work
through [bold cyan]{}[/][/]".format(qp), style="bold white")

      else:
            console.rule("\n[bold blue]Great! Please wait while we work
through [bold cyan]{} and {}[/][/]".format(qp, ms), style="bold white")

      start(input_file_structure, qp, ms, output_format)

      if ms != None:
            progress.console.log(f"Processed {qp} and {ms}")
      else:
            progress.console.log(f"Processed {qp}")

      console.print()
      console.rule("[bold blue]Done![/]", style="bold white")
      console.print("Please check your past paper file location again,
there should be a new folder containing all the cropped and sorted
individual images of questions/answers!\n\n")
      console.rule("[bold blue]Terminating", style = "bold red")
```

init.py

```
import os, sys, shutil

def start():
    if os.path.exists("QPResults"):
       shutil.rmtree("QPResults")
    if os.path.exists("MSResults"):
       shutil.rmtree("MSResults")

    os.system("pip install -r requirements.txt")
    os.system("clear")
    system = sys.platform
    os_name = os.name
    if system == "darwin" or system == "linux":
       return True
    elif system == "win32":
       return False

start()
```

cleaning.py

```
import os, shutil

def clean_up(out_type):
    if os.path.exists("{}Results/Q0.jpg".format(out_type)):
       os.remove("{}Results/Q0.jpg".format(out_type))
    shutil.rmtree("og")
    if out_type == "QP":
       os.remove("complete.jpg")
       os.remove("left_long.jpg")
       shutil.rmtree("cropped")
       shutil.rmtree("left")
    else:
       shutil.rmtree("table")
```

crop_final.py

```
#crop based on question number (top) and last score (bottom)
import os, cv2, pytesseract
from PIL import Image
import cv2
import pytesseract

def output_crop(q_s_dict, output_type):
    code_image = cv2.imread("og/out3.jpg")
    paper_code_roi = code_image[2198:2339, 516:1151].copy()
    custom_config = "--oem 3 --psm 6"
    paper_code = pytesseract.image_to_string(paper_code_roi, config =
custom_config, lang = "eng")
    if "_" in paper_code:
        paper_code = paper_code.split("_")
        paper_code.insert(3, "M")
        paper_code.insert(4, "J")
        paper_code[-2] = paper_code[-2][-2]
        paper_code = paper_code[1:-2]
    elif "/" in paper_code:
        paper_code = paper_code.split("/")

    copy_dict = q_s_dict.copy()
    for key in copy_dict:
        copy_dict[key] = list(copy_dict[key][-1])

    if not os.path.exists("QPResults"):
        os.makedirs("QPResults")

    img = Image.open("complete.jpg")
    width, height = img.size
    area = (0, 0, 0, 0)
    for question in copy_dict:
       new_area = (0, area[3], width,
int(copy_dict[question][1]+copy_dict[question][3])+50)
       result = img.crop(new_area)
       result.save("QPResults/"+"_".join(paper_code)+"_Q" + question +
".{}".format(output_type))
       area = new_area
    return paper_code
```

crop_ms.py

```
import cv2, os

def sorting_ms_dict(question_dict):
    pages_depth = []
    depth = lambda L: isinstance(L, list) and max(map(depth, L))+1
    for key in question_dict:
       page = depth(question_dict[key])
       pages_depth.append(page)
    for page in range(1, len(pages_depth)+1):
       if pages_depth[page-1] == 2:
             question_dict[page] = question_dict[page][-1]
       else:
             question_dict[page] =
question_dict[page][-pages_depth[page-1]+1:]
             for x in range(1, pages_depth[page-1]-1):
                    question_dict[page][x] = question_dict[page][x][-1]
    return (question_dict, pages_depth)

def cropping_ms(paper_code, x, y, x2, y2, question_dict, pages_depth,
output_format):
    if not os.path.exists("MSResults"):
       os.makedirs("MSResults")
    for key in question_dict:
       if pages_depth[key-1] > 2:
             for page in range(1, len(question_dict[key])):
                    image = cv2.imread(question_dict[key][page][-1])
                    height, width, _ = image.shape
                    temp_y = y2-y
                    wo_header = image[temp_y:height, 0:width]
                    cv2.imwrite(question_dict[key][page][-1], wo_header)
             final = []
```

```
            for page in range(len(question_dict[key])):
                  final.append(question_dict[key][page][-1])
            image = cv2.vconcat([cv2.imread(img) for img in final])
      else:
            image = cv2.imread(question_dict[key][-1])
      cv2.imwrite("MSResults/{}.{}".format("_".join(paper_code)+"_Q" +
str(key), output_format), image)
```

crop_table.py

```
import os, cv2, re
import numpy as np
import pytesseract
from pytesseract import Output

def check_GMP(images):
    for i in range(1, len(images)):
       image_path = "og/out{}.jpg".format(i)
       image = cv2.imread(image_path)
       data = pytesseract.image_to_string(image, config = "--oem 3 --psm
6", lang = "eng")
       if "Generic Marking Principles" not in data:
             return i
             break

def find_table(num):
    boxes = []
    image_path = "og/out{}.jpg".format(num)
    image = cv2.imread(image_path)
    gray_scale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    th1, img_bin = cv2.threshold(gray_scale, 150, 255, cv2.THRESH_BINARY)
    img_bin =~img_bin
    line_min_width = 50
    kernel_h = np.ones((1, line_min_width), np.uint8)
    kernel_v = np.ones((line_min_width,1), np.uint8)
    kernel = np.ones((20,1), np.uint8)
    d_im = cv2.dilate(img_bin, kernel, iterations=1)
    e_im = cv2.erode(d_im, kernel, iterations=1)
    img_bin_h = cv2.morphologyEx(e_im, cv2.MORPH_OPEN, kernel_h)
    img_bin_v = cv2.morphologyEx(e_im, cv2.MORPH_OPEN, kernel_v)
    img_bin_final = img_bin_h | img_bin_v
    _, _, stats, _ = cv2.connectedComponentsWithStats(~img_bin_final,
connectivity=4, ltype=cv2.CV_32S)
```

```
    for x,y,w,h,area in stats[2:]:
       roi = image[y:y+h, x:x+w].copy()
       custom_config = r"--oem 3 --psm 4"
       data = pytesseract.image_to_data(roi, 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("(Question?|Answer?|Marks?)")
       headings = [(text) for left, top, width, height, text in
list(zip(*data.values())) if pattern.match(text)]
       boxes.append(((x, y, w, h, area), headings))
    return boxes


def sort_table(count_temp, boxes):
    new_count_temp = count_temp
    for stats, text in boxes:
       if text == ["Question"]:
             new_count_temp +=1
    box_dict = dict.fromkeys(range(count_temp, new_count_temp))
    box_zip, end_zip = [], []
    col3_x, col3_w = 0, 0
    for (grid, text) in boxes:
       if text == ["Marks"]:
             col3_x, col3_w = grid[0], grid[2]
       if text == ["Question"]:
             col3_x, col3_w = 0, 0
             box_zip.append([])
             end_zip.append([])
             box_zip[-1].append((grid, text))
       else:
             if grid[0] == col3_x and text != ["Marks"]:
                   end_zip[-1].append((grid, text))
    i = 0
    for key in box_dict:
       box_dict[key] = box_zip[i]
       box_dict[key].append(end_zip[i])
       i+=1
    count_temp = new_count_temp
    return(count_temp, box_dict)
```

```
def cropping(page, box_dict, images):
    newpath = r"table"
    if not os.path.exists(newpath):
       os.makedirs(newpath)
    for key in box_dict:
       new_image_path = "table/table{}.jpg".format(key)
       og_page = cv2.imread(images[page])
       top_left_x = box_dict[key][0][0][0]
       top_left_y = box_dict[key][0][0][1]
       bottom_right_x =
box_dict[key][-1][-1][0][0]+box_dict[key][-1][-1][0][2]
       bottom_right_y =
box_dict[key][-1][-1][0][1]+box_dict[key][-1][-1][0][3]
       new_image = og_page[top_left_y:bottom_right_y,
top_left_x:bottom_right_x]
       cv2.imwrite("{}".format(new_image_path), new_image)
    return box_dict[key][0][0][2]

def save_header(box_dict, images):
    og_page = cv2.imread(images[3])
    for key in box_dict:
       top_left_x = box_dict[key][0][0][0]
       top_left_y = box_dict[key][0][0][1]
       bottom_right_x = box_dict[key][1][0][0][0]+box_dict[key][1][0][0][2]
       bottom_right_y = top_left_y+box_dict[key][0][0][3]
    header = og_page[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
    cv2.imwrite("table/header.jpg", header)
    return (top_left_x, top_left_y, bottom_right_x, bottom_right_y)
```

input_manipulation.py

```
from PIL import Image
from pytesseract import Output
import numpy as np
import cv2, os, re, pytesseract

def tessa(images, count):
    crop_bottom = [None]
    for i in range(1, count):
       image = cv2.imread(images[i])
       custom_config = r'-c tessedit_char_whitelist=[]1234567890 --psm 11'
       data = pytesseract.image_to_data(image, output_type = Output.DICT,
config=custom_config)
```

```
       for data_type in list(data):
             if data_type not in ["text", "height", "width", "top",
"left"]:
                   data.pop(data_type, 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)]
       if len(score_result) == 0:
             crop_bottom.append(None)
             pass
       else:
             final_score = score_result[-1]
             crop_bottom.append(final_score[1]+final_score[3]+35)
    return crop_bottom, count

def cropping(images, count, crop_bottom):
    cropped, left = [], []
    for i in range(1, count):
       img = cv2.imread(images[i])
       if crop_bottom[i] == None:
             pass
       else:
             cropped_img = img[141:crop_bottom[i], 0:1654].copy()
             if not os.path.exists("cropped"):
                   os.makedirs("cropped")
             cv2.imwrite("cropped/cropped" + str(i) + ".jpg", cropped_img)
             cropped.append('cropped/cropped' + str(i) + '.jpg')
             left_tab = img[141:crop_bottom[i], 0:201].copy()
             if not os.path.exists("left"):
                   os.makedirs("left")
             cv2.imwrite('left/left_tab' + str(i) + '.jpg', left_tab)
             left.append('left/left_tab' + str(i) + '.jpg')
    return cropped, left

def merge(cropped, left):
    merged = cv2.vconcat([cv2.imread(img) for img in cropped])
    cv2.imwrite("complete.jpg", merged)
    merged = cv2.vconcat([cv2.imread(img) for img in left])
    cv2.imwrite("left_long.jpg", merged)
```

make_folder.py

```
from pdf2image import convert_from_path
```

```python
import sys, os

def io_mac(file_name):
    path_file = str(file_name)
    pages = convert_from_path(file_name)
    images = []
    count = 0
    for page in pages:
       newpath = r"og"
       if not os.path.exists(newpath):
             os.makedirs(newpath)
       page.save("og/out" + str(count) + ".jpg", "JPEG")
       images.append("og/out" + str(count) + ".jpg")
       count += 1
    return (images, count)

def io_win(file_name):
    path_file = str(sys.argv[1])
    pages = convert_from_path(path_file)
```

markscheme.py

```python
import make_folder, crop_table, read_question_ms, crop_ms, cleaning
import numpy as np

def markscheme_program(ms, paper_code, output_type):
    (images, count) = make_folder.io_mac(ms)
    ms_question_dict = {}
    count_temp = 1
    ms_start = crop_table.check_GMP(images)
    for ms_count in range(ms_start, len(images)):
       boxes = crop_table.find_table(ms_count)
       (count_temp, box_dict) = crop_table.sort_table(count_temp, boxes)
       ms_question_dict.update(box_dict)
       images = [images[index] for index in range(count)]
       width = crop_table.cropping(ms_count, box_dict, np.array(images))
    (x, y, x2, y2) = crop_table.save_header(box_dict, images)
    question_dict = {}
    for ms_2_count in range(1,count_temp):
       question_result = read_question_ms.question_OCR(width, ms_2_count)
       question_dict = read_question_ms.sort_questions(question_dict,
question_result)
    (question_dict, pages_depth) = crop_ms.sorting_ms_dict(question_dict)
    crop_ms.cropping_ms(paper_code, x, y, x2, y2, question_dict,
pages_depth, output_type)
    cleaning.clean_up("MS")
```

qs_mainpulation.py [*sic*]

```
def sort_qs(question_result, score_result):
    q_s_dict = {}
    for q in range(len(question_result)-1):
       for s in range(len(score_result)):
             if question_result[q][1] < score_result[s][1] <
question_result[q+1][1]:
                   if question_result[q][4] in q_s_dict:

q_s_dict[question_result[q][4]].append(score_result[s])
                   else:
                         q_s_dict[question_result[q][4]]= [score_result[s]]
             elif question_result[q+1][1] == question_result[-1][1] and
question_result[-1][1] < score_result[s][1]:
                   if question_result[-1][4] in q_s_dict:

q_s_dict[question_result[-1][4]].append(score_result[s])
                   else:
                         q_s_dict[question_result[-1][4]]=
[score_result[s]]
    return q_s_dict
```

question.py

```
import make_folder, input_manipulation, question_score, qs_mainpulation,
crop_final, cleaning

def question_program(qp, output_type):
    (images, count) = make_folder.io_mac(qp)
    (crop_bottom, count) = input_manipulation.tessa(images, count)
    (cropped, left) = input_manipulation.cropping(images, count,
crop_bottom)
    input_manipulation.merge(cropped, left)
    (question_result, score_result) = question_score.find_q_s()
    q_s_dict = qs_mainpulation.sort_qs(question_result, score_result)
    paper_code = crop_final.output_crop(q_s_dict, "QP", output_type)
    cleaning.clean_up("QP")
    return paper_code
```

question_score.py

```
import cv2
from pytesseract import Output
import pytesseract, re

def question_OCR(img):
    height, width, channels = img.shape
    question_roi = img[0:height, 0:180].copy()
    custom_config = r"--oem 3 --psm 6 outputbase digits"
    data = pytesseract.image_to_data(question_roi, 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)
    question_result = [(left, top, width, height, text) for left, top,
width, height, text in zip(*data.values()) if text.isdigit()]
    return question_result

def score_OCR(img):
    height, width, channels = img.shape
    score_roi = img[0:height, 1474:1654].copy()
    custom_config = r'-c tessedit_char_whitelist=[]1234567890 --psm 11'
    data = pytesseract.image_to_data(score_roi, 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)]
    return score_result

def find_q_s():
    img = cv2.imread('complete.jpg')
    question_result = question_OCR(img)
    score_result = score_OCR(img)
    return (question_result, score_result)
```

read_question_ms.py

```
import pytesseract,re, cv2
from pytesseract import Output
```

```
def question_OCR(width, i):
    img_path = "table/table{}.jpg".format(i)
    img = cv2.imread(img_path)
    height, _, _ = img.shape
    question_roi = img[0:height, 0:width].copy()
    custom_config = r"--oem 3 --psm 4"
    data = pytesseract.image_to_data(question_roi, 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("^[0-9]")
    question_result = [[left, top, width, height, text] for left, top,
width, height, text in zip(*data.values()) if pattern.match(text)]
    for tup in question_result:
       tup.append(img_path)
    return question_result

def sort_questions(question_dict, question_result):
    key = int(question_result[0][4].split("(")[0])
    if key not in question_dict:
       question_dict[key] = question_result
    else:
       question_dict[key].append(question_result)
    return question_dict
```

score.py

```
#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)]
```