prot_scan

January 24, 2023

[1]:
```
import csv
import PyPDF2 as pypdf
import json
import os
from pdf2image import convert_from_path
import numpy as np
import cv2
```

1 Detect Statement

First lets take a look at the data extracted with pyPDF2 from the statement:

[2]:
```
var = pypdf.PdfFileReader(open('pdf/statement_11111.pdf', 'rb')).getFields()
print(var)
```

```
{'1:case:basic-adittion:1,1': {'/T': '1:case:basic-adittion:1,1', '/FT': '/Btn',
'/V': '/Off', '/Rect': [89.867, 563.517, 102.583, 576.233]}, '1:case:basic-
adittion:1,2': {'/T': '1:case:basic-adittion:1,2', '/FT': '/Btn', '/V': '/Off',
'/Rect': [111.549, 563.517, 124.265, 576.233]}, '1:case:basic-adittion:1,3':
{'/T': '1:case:basic-adittion:1,3', '/FT': '/Btn', '/V': '/Off', '/Rect':
[133.232, 563.517, 145.948, 576.233]}, '1:case:basic-adittion:1,4': {'/T':
'1:case:basic-adittion:1,4', '/FT': '/Btn', '/V': '/Off', '/Rect': [154.914,
563.517, 167.631, 576.233]}, '1:case:basic-adittion:1,5': {'/T': '1:case:basic-
adittion:1,5', '/FT': '/Btn', '/V': '/Off', '/Rect': [176.597, 563.517, 189.313,
576.233]}, '1:case:basic-adittion:1,6': {'/T': '1:case:basic-adittion:1,6',
'/FT': '/Btn', '/V': '/Off', '/Rect': [198.28, 563.517, 210.996, 576.233]},
'1:case:basic-adittion:1,7': {'/T': '1:case:basic-adittion:1,7', '/FT': '/Btn',
'/V': '/Off', '/Rect': [219.962, 563.517, 232.678, 576.233]}, '1:case:basic-
adittion:1,8': {'/T': '1:case:basic-adittion:1,8', '/FT': '/Btn', '/V': '/Off',
'/Rect': [241.645, 563.517, 254.361, 576.233]}, '1:case:basic-adittion:1,9':
{'/T': '1:case:basic-adittion:1,9', '/FT': '/Btn', '/V': '/Off', '/Rect':
[263.327, 563.517, 276.044, 576.233]}, '1:case:basic-adittion:1,10': {'/T':
'1:case:basic-adittion:1,10', '/FT': '/Btn', '/V': '/Off', '/Rect': [285.01,
563.517, 297.726, 576.233]}, '1:case:basic-adittion:1,11': {'/T': '1:case:basic-
adittion:1,11', '/FT': '/Btn', '/V': '/Off', '/Rect': [89.867, 547.272, 102.583,
559.988]}, '1:case:basic-adittion:1,12': {'/T': '1:case:basic-adittion:1,12',
'/FT': '/Btn', '/V': '/Off', '/Rect': [111.549, 547.272, 124.265, 559.988]},
```

'1:case:basic-adittion:1,13': {'/T': '1:case:basic-adittion:1,13', '/FT':
'/Btn', '/V': '/Off', '/Rect': [133.232, 547.272, 145.948, 559.988]},
'1:case:basic-adittion:1,14': {'/T': '1:case:basic-adittion:1,14', '/FT':
'/Btn', '/V': '/Off', '/Rect': [154.914, 547.272, 167.631, 559.988]},
'1:case:basic-adittion:1,15': {'/T': '1:case:basic-adittion:1,15', '/FT':
'/Btn', '/V': '/Off', '/Rect': [176.597, 547.272, 189.313, 559.988]},
'1:case:basic-adittion:1,16': {'/T': '1:case:basic-adittion:1,16', '/FT':
'/Btn', '/V': '/Off', '/Rect': [198.28, 547.272, 210.996, 559.988]},
'1:case:basic-adittion:1,17': {'/T': '1:case:basic-adittion:1,17', '/FT':
'/Btn', '/V': '/Off', '/Rect': [219.962, 547.272, 232.678, 559.988]},
'1:case:basic-adittion:1,18': {'/T': '1:case:basic-adittion:1,18', '/FT':
'/Btn', '/V': '/Off', '/Rect': [241.645, 547.272, 254.361, 559.988]},
'1:case:basic-adittion:1,19': {'/T': '1:case:basic-adittion:1,19', '/FT':
'/Btn', '/V': '/Off', '/Rect': [263.327, 547.272, 276.044, 559.988]},
'1:case:basic-adittion:1,20': {'/T': '1:case:basic-adittion:1,20', '/FT':
'/Btn', '/V': '/Off', '/Rect': [285.01, 547.272, 297.726, 559.988]},
'1:case:derivativesin:2,4': {'/T': '1:case:derivativesin:2,4', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 480.635, 105.961, 493.352]},
'1:case:derivativesin:2,2': {'/T': '1:case:derivativesin:2,2', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 464.695, 105.961, 477.411]},
'1:case:derivativesin:2,3': {'/T': '1:case:derivativesin:2,3', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 448.755, 105.961, 461.471]},
'1:case:derivativesin:2,1': {'/T': '1:case:derivativesin:2,1', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 432.815, 105.961, 445.531]}}

The extracted data is a dictionary, and the keys are a string which contains the exercise name and two numbers. Those two numbers seem to be the exercise number and box number, but when looking at the boxes in the second exercise they appear to be out of order.

The value of each key is another dictionary, with the following keys: '/T': contains the same string as the original key '/FT': all have the value '/Btn '/V': stores whether the box is marked. As this is the statement pdf, none of them are marked '/Rect': has the coordinates of the box

1.0.1 JSON format for statement

[3]:
```
s_statement = {"exam": "Multiple choice",
               "num_boxes": 4,
               "exercises": [
                   {"exercise": "basic-adittion",
                    "checkboxes": [
                        {
                            "checkbox": "0,0",
                            "cords": [
                                89.867,
                                563.517,
                                12.715999999999994,
                                12.715999999999894
                            ]
```

```
                        },
                        {
                            "checkbox": "0,1",
                            "cords": [
                                111.549,
                                563.517,
                                12.715999999999994,
                                12.715999999999894
                            ]
                        }
                    ]},
                   {"exercise": "derivativesin",
                    "checkboxes": [
                        {
                            "checkbox": "1,0",
                            "cords": [
                                93.245,
                                480.635,
                                12.715999999999994,
                                12.716999999999985
                            ]
                        },
                        {
                            "checkbox": "1,1",
                            "cords": [
                                93.245,
                                464.695,
                                12.715999999999994,
                                12.716000000000008
                            ]
                        }
                    ]}
                ]}
s_statement
```

[3]:
```
{'exam': 'Multiple choice',
 'num_boxes': 4,
 'exercises': [{'exercise': 'basic-adittion',
   'checkboxes': [{'checkbox': '0,0',
     'cords': [89.867, 563.517, 12.715999999999994, 12.715999999999894]},
    {'checkbox': '0,1',
     'cords': [111.549, 563.517, 12.715999999999994, 12.715999999999894]}]},
  {'exercise': 'derivativesin',
   'checkboxes': [{'checkbox': '1,0',
     'cords': [93.245, 480.635, 12.715999999999994, 12.716999999999985]},
    {'checkbox': '1,1',
     'cords': [93.245, 464.695, 12.715999999999994, 12.716000000000008]}]}]}
```

Function that, given a pdf file with the statement and the exam id, detects the statement structure, exercises and checkboxes:

[4]:
```
def detect_st(st_file, id_exam):
    # open the pdf file with the exam statement ?- check whether file exists
    # get the checkboxes from the modified pyPDF2 version
    var = pypdf.PdfFileReader(st_file).getFields()

    # process and store the checkbox information on the json files
    #   data stores the information to be stored
    #   count_ex iterates the exercises
    #   count_box iterates the checkboxes for each exercise
    data = {'exam': id_exam,
            'num_boxes': var.__len__(),
            'exercises' : []}

    count_ex = -1
    count_box = -1
    for box in var:                 # example of content in box: 1:case:
↪basic-adittion:1,1
        exercise = box.split(sep=':')[2]
        # if it is the first box to be read, or if the new box being read is␣
↪from a next exercise --> append exercise
        if not data['exercises'] or (data['exercises'][count_ex]['exercise'] !=␣
↪exercise):
            count_ex = count_ex + 1
            count_box = 0
            data['exercises'].append(
                {
                    'exercise': exercise,
                    'checkboxes': []
                }
            )
        # append checkbox
        count_box = count_box + 1
        data['exercises'][count_ex]['checkboxes'].append(
            {
                'checkbox' : str(count_ex) + ',' + str(count_box),
                'cords' : [ float(var[box]['/Rect'][0]),
                            float(var[box]['/Rect'][1]),
                            round(float(var[box]['/Rect'][2]) -␣
↪float(var[box]['/Rect'][0]), 3),
                            round(float(var[box]['/Rect'][3]) -␣
↪float(var[box]['/Rect'][1]), 3) ]
            })
    # write the JSON file
    jf_name = 'json/st_' + id_exam + '.json'
```

```
    os.makedirs(os.path.dirname(jf_name), exist_ok=True)
    with open(jf_name, 'w') as outfile:
        json.dump(data, outfile, indent=4)
```

test behaviour:

[5]:
```
detect_st('pdf/statement_11111.pdf', "Test")
```

2 Detect Solution

First let's take a look at the data extracted with pyPDF2 from the solution:

[6]:
```
var = pypdf.PdfFileReader(open('pdf/solution_11111.pdf', 'rb')).getFields()
print(var)
```


```
{'1:case:basic-adittion:1,1': {'/T': '1:case:basic-adittion:1,1', '/FT': '/Btn',
'/V': '/Off', '/Rect': [89.867, 581.13, 102.583, 593.846]}, '1:case:basic-
adittion:1,2': {'/T': '1:case:basic-adittion:1,2', '/FT': '/Btn', '/V': '/Yes',
'/Rect': [111.549, 581.13, 124.265, 593.846]}, '1:case:basic-adittion:1,3':
{'/T': '1:case:basic-adittion:1,3', '/FT': '/Btn', '/V': '/Off', '/Rect':
[133.232, 581.13, 145.948, 593.846]}, '1:case:basic-adittion:1,4': {'/T':
'1:case:basic-adittion:1,4', '/FT': '/Btn', '/V': '/Off', '/Rect': [154.914,
581.13, 167.631, 593.846]}, '1:case:basic-adittion:1,5': {'/T': '1:case:basic-
adittion:1,5', '/FT': '/Btn', '/V': '/Off', '/Rect': [176.597, 581.13, 189.313,
593.846]}, '1:case:basic-adittion:1,6': {'/T': '1:case:basic-adittion:1,6',
'/FT': '/Btn', '/V': '/Off', '/Rect': [198.28, 581.13, 210.996, 593.846]},
'1:case:basic-adittion:1,7': {'/T': '1:case:basic-adittion:1,7', '/FT': '/Btn',
'/V': '/Off', '/Rect': [219.962, 581.13, 232.678, 593.846]}, '1:case:basic-
adittion:1,8': {'/T': '1:case:basic-adittion:1,8', '/FT': '/Btn', '/V': '/Off',
'/Rect': [241.645, 581.13, 254.361, 593.846]}, '1:case:basic-adittion:1,9':
{'/T': '1:case:basic-adittion:1,9', '/FT': '/Btn', '/V': '/Off', '/Rect':
[263.327, 581.13, 276.044, 593.846]}, '1:case:basic-adittion:1,10': {'/T':
'1:case:basic-adittion:1,10', '/FT': '/Btn', '/V': '/Off', '/Rect': [285.01,
581.13, 297.726, 593.846]}, '1:case:basic-adittion:1,11': {'/T': '1:case:basic-
adittion:1,11', '/FT': '/Btn', '/V': '/Off', '/Rect': [89.867, 564.885, 102.583,
577.602]}, '1:case:basic-adittion:1,12': {'/T': '1:case:basic-adittion:1,12',
'/FT': '/Btn', '/V': '/Off', '/Rect': [111.549, 564.885, 124.265, 577.602]},
'1:case:basic-adittion:1,13': {'/T': '1:case:basic-adittion:1,13', '/FT':
'/Btn', '/V': '/Off', '/Rect': [133.232, 564.885, 145.948, 577.602]},
'1:case:basic-adittion:1,14': {'/T': '1:case:basic-adittion:1,14', '/FT':
'/Btn', '/V': '/Yes', '/Rect': [154.914, 564.885, 167.631, 577.602]},
'1:case:basic-adittion:1,15': {'/T': '1:case:basic-adittion:1,15', '/FT':
'/Btn', '/V': '/Off', '/Rect': [176.597, 564.885, 189.313, 577.602]},
'1:case:basic-adittion:1,16': {'/T': '1:case:basic-adittion:1,16', '/FT':
'/Btn', '/V': '/Off', '/Rect': [198.28, 564.885, 210.996, 577.602]},
'1:case:basic-adittion:1,17': {'/T': '1:case:basic-adittion:1,17', '/FT':
'/Btn', '/V': '/Off', '/Rect': [219.962, 564.885, 232.678, 577.602]},
```

```
'1:case:basic-adittion:1,18': {'/T': '1:case:basic-adittion:1,18', '/FT':
'/Btn', '/V': '/Off', '/Rect': [241.645, 564.885, 254.361, 577.602]},
'1:case:basic-adittion:1,19': {'/T': '1:case:basic-adittion:1,19', '/FT':
'/Btn', '/V': '/Off', '/Rect': [263.327, 564.885, 276.044, 577.602]},
'1:case:basic-adittion:1,20': {'/T': '1:case:basic-adittion:1,20', '/FT':
'/Btn', '/V': '/Off', '/Rect': [285.01, 564.885, 297.726, 577.602]},
'1:case:derivativesin:2,4': {'/T': '1:case:derivativesin:2,4', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 498.248, 105.961, 510.965]},
'1:case:derivativesin:2,2': {'/T': '1:case:derivativesin:2,2', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 482.308, 105.961, 495.024]},
'1:case:derivativesin:2,3': {'/T': '1:case:derivativesin:2,3', '/FT': '/Btn',
'/V': '/Off', '/Rect': [93.245, 466.368, 105.961, 479.084]},
'1:case:derivativesin:2,1': {'/T': '1:case:derivativesin:2,1', '/FT': '/Btn',
'/V': '/Yes', '/Rect': [93.245, 450.428, 105.961, 463.144]}}
```

2.0.1 JSON format for solution

Following the same pattern of statement, we add data for the variant id and which checkboxes are marked on each exercise

[7]:
```
s_solution = {"exam": "Multiple choice",
              "variant": 11111,                                  # new key to save␣
 ↪the variant id
               "num_boxes": 4,
               "exercises": [
                   {"exercise": "basic-adittion",
                    "checkboxes": [
                        {
                            "checkbox": "0,0",
                            "cords": [
                                89.867,
                                563.517,
                                12.715999999999994,
                                12.715999999999894
                            ]
                        },
                        {
                            "checkbox": "0,1",
                            "cords": [
                                111.549,
                                563.517,
                                12.715999999999994,
                                12.715999999999894
                            ]
                        }],
                    "sol_marked": ["0,1"] },                     # new array to␣
 ↪store the marked checkboxes in the solution
                   {"exercise": "derivativesin",
```

```
                    "checkboxes": [
                        {
                            "checkbox": "1,0",
                            "cords": [
                                93.245,
                                480.635,
                                12.715999999999994,
                                12.716999999999985
                            ]
                        },
                        {
                            "checkbox": "1,1",
                            "cords": [
                                93.245,
                                464.695,
                                12.715999999999994,
                                12.716000000000008
                            ]
                        }],
                    "sol_marked": ["0,0"] },
                ]}
s_solution
```

[7]:
```
{'exam': 'Multiple choice',
 'variant': 11111,
 'num_boxes': 4,
 'exercises': [{'exercise': 'basic-adittion',
   'checkboxes': [{'checkbox': '0,0',
     'cords': [89.867, 563.517, 12.715999999999994, 12.715999999999894]},
    {'checkbox': '0,1',
     'cords': [111.549, 563.517, 12.715999999999994, 12.715999999999894]}],
   'sol_marked': ['0,1']},
  {'exercise': 'derivativesin',
   'checkboxes': [{'checkbox': '1,0',
     'cords': [93.245, 480.635, 12.715999999999994, 12.716999999999985]},
    {'checkbox': '1,1',
     'cords': [93.245, 464.695, 12.715999999999994, 12.716000000000008]}],
   'sol_marked': ['0,0']}]}
```

[8]:
```
def detect_sol(sol_file, st_json, id_variant):
    # open the pdf file with the exam solution ?- check whether file exists
    # get the checkboxes from the modified pyPDF2 version
    var = pypdf.PdfFileReader(sol_file).getFields()

    # open and read the statement JSON
    with open(st_json) as json_file:
        data = json.load(json_file)
```

```
    # process the solution pdf into the dictionary
    # check if it has the same number of checkboxes? or assume correct?
    data['variant'] = id_variant
    count_ex = -1
    count_box = -1
    st_exercise = ''
    for box in var:                 # example of content in box: 1:case:
↪basic-adittion:1,1
        sol_exercise = box.split(sep=':')[2]
        # if it is the first box to be read, or if the current box is in a new␣
↪exercise --> update st_exercise and add 'sol_marked' key to dict
        # check if the current box is in the same exercise. assumes the␣
↪exercise names are the same and in the same order
        if not sol_exercise == st_exercise:
            count_ex = count_ex + 1
            count_box = -1
            st_exercise = data['exercises'][count_ex]['exercise']
            data['exercises'][count_ex]['sol_marked'] = []
        # check if the current box is marked. appends if marked
        count_box = count_box + 1
        if var[box]['/V'] == '/Yes':
            data['exercises'][count_ex]['sol_marked'].append(str(count_ex) +␣
↪',' + str(count_box))

    # write the JSON file
    jf_name = 'json/sol_' + id_variant+ '.json'
    os.makedirs(os.path.dirname(jf_name), exist_ok=True)
    with open(jf_name, 'w') as outfile:
        json.dump(data, outfile, indent=4)
```

[9]:
```
detect_sol('pdf/solution_11111.pdf','json/st_Test.json', '11111')
```

3 Analyse Scan

First we need to get the checkbox info from the pdf with OpenCV Based on https://towardsdatascience.com/checkbox-table-cell-detection-using-opencv-python-332c57d25171

[10]:
```
def find_cboxes_img(png_file):
    img = cv2.imread(png_file)
    gray_scale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    th1, img_bin = cv2.threshold(gray_scale, 150, 225, cv2.THRESH_BINARY)
    img_bin = ~img_bin

    # set min width of lines for the rectangle: 20 pixels
    line_min_width = 20
```

```
    kernel_h = np.ones((1, line_min_width), np.uint8)
    kernel_v = np.ones((line_min_width, 1), np.uint8)

    # find horizontal and vertical lines and join them
    img_bin_h = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN, kernel_h)
    img_bin_v = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN, kernel_v)
    img_bin_final = img_bin_h | img_bin_v

    # find conected lines that form a box
    _, labels, stats, _ = cv2.connectedComponentsWithStats(~img_bin_final,␣
↪connectivity=8, ltype=cv2.CV_32S)
    # stats: (x, y, w, h, a) * num_boxes
    # first two rows of stats are the background and residue pixel bounding␣
↪boxes, which we dont need
    return stats[2:]
```

However, this function uses a png file as an input, and we have a pdf We have to divide it into pages, export those pages to png, and process each page

[11]:
```
def find_cboxes(pdf_file, id_student):
    boxes = []
    # export the pdf's pages to png with 250 dots per inch (dpi)
    pages = convert_from_path(pdf_file, 250)
    # create the directory png
    if not os.path.exists('png'):
        os.mkdir('png')
    # iterate the pages
    for i, page in enumerate(pages):
        # save each page in a png file
        im_file = 'png/' + id_student + '_page' + str(i) + '.png'
        page.save(im_file)

        # add the checkboxes to the list
        boxes.append(find_cboxes_img(im_file))
    return boxes
```

Now we have the checkboxes. The function returns a list of arrays like [x, y, w, h, a]. These values are the coordinates (x, y), the width and height of the boxes, and an area. This area is close to the area of the square when the box is not filled, and smaller when it is filled

[12]:
```
cboxes = find_cboxes('pdf/scanned_11111.pdf', '11111')
cboxes
```

[12]:
```
[array([[ 324,  914,   37,   37,  704],
        [ 399,  915,   37,   37, 1325],
        [ 475,  916,   36,   37, 1318],
        [ 550,  918,   36,   36, 1296],
```

```
[ 625,  919,   37,   37, 1326],
[ 700,  920,   37,   37, 1324],
[ 775,  922,   37,   36, 1302],
[ 850,  923,   37,   37, 1307],
[ 926,  924,   37,   37, 1347],
[1001,  926,   37,   36, 1310],
[ 323,  970,   37,   37, 1317],
[ 398,  971,   37,   37, 1332],
[ 474,  973,   36,   36, 1296],
[ 549,  974,   36,   37,  688],
[ 624,  975,   37,   37, 1326],
[ 699,  977,   37,   36, 1311],
[ 775,  978,   36,   37, 1299],
[ 850,  979,   36,   37, 1298],
[ 925,  981,   37,   36, 1306],
[1000,  982,   37,   37, 1309],
[ 331, 1202,   37,   36, 1309],
[ 330, 1257,   37,   37, 1340],
[ 329, 1312,   37,   37, 1326],
[ 328, 1367,   37,   37,  745]], dtype=int32)]
```

3.0.1 JSON format for scan

Following the same pattern of statement and solution, we add data for the student id and which checkboxes are marked on each exercise

[13]:
```
s_scan = {"exam": "Multiple choice",
          "student": 11111,                                    # new key to store␣
 ↪the student id
          "num_boxes": 4,
          "exercises": [
              {"exercise": "basic-adittion",
               "checkboxes": [
                   {
                       "checkbox": "0,0",
                       "cords": [
                           89.867,
                           563.517,
                           12.715999999999994,
                           12.715999999999894
                       ]
                   },
                   {
                       "checkbox": "0,1",
                       "cords": [
                           111.549,
                           563.517,
```

```
                        12.715999999999994,
                        12.715999999999894
                    ]
                }],
             "scan_marked": ["0,1"] },                    # new array to␣
 ↪store the marked checkboxes in the scanned exam
            {"exercise": "derivativesin",
             "checkboxes": [
                 {
                     "checkbox": "1,0",
                     "cords": [
                         93.245,
                         480.635,
                         12.715999999999994,
                         12.716999999999985
                     ]
                 },
                 {
                     "checkbox": "1,1",
                     "cords": [
                         93.245,
                         464.695,
                         12.715999999999994,
                         12.716000000000008
                     ]
                 }],
             "scan_marked": ["0,1"] },
        ]}
s_scan
```

[13]:
```
{'exam': 'Multiple choice',
 'student': 11111,
 'num_boxes': 4,
 'exercises': [{'exercise': 'basic-adittion',
   'checkboxes': [{'checkbox': '0,0',
     'cords': [89.867, 563.517, 12.715999999999994, 12.715999999999894]},
    {'checkbox': '0,1',
     'cords': [111.549, 563.517, 12.715999999999994, 12.715999999999894]}],
   'scan_marked': ['0,1']},
  {'exercise': 'derivativesin',
   'checkboxes': [{'checkbox': '1,0',
     'cords': [93.245, 480.635, 12.715999999999994, 12.716999999999985]},
    {'checkbox': '1,1',
     'cords': [93.245, 464.695, 12.715999999999994, 12.716000000000008]}],
   'scan_marked': ['0,1']}]}
```

[14]:
```
for i in range(s_scan['exercises'].__len__()):
    print(s_scan['exercises'][i])
```

```
{'exercise': 'basic-adittion', 'checkboxes': [{'checkbox': '0,0', 'cords':
[89.867, 563.517, 12.715999999999994, 12.715999999999894]}, {'checkbox': '0,1',
'cords': [111.549, 563.517, 12.715999999999994, 12.715999999999894]}],
'scan_marked': ['0,1']}
{'exercise': 'derivativesin', 'checkboxes': [{'checkbox': '1,0', 'cords':
[93.245, 480.635, 12.715999999999994, 12.716999999999985]}, {'checkbox': '1,1',
'cords': [93.245, 464.695, 12.715999999999994, 12.716000000000008]}],
'scan_marked': ['0,1']}
```

Now we need to make a function that receives the scanned pdf, the json file with the statement data, and the student id; and generates a json file with the structure mentioned. We already have a function that returns the array with information on the checkboxes, now we need to transform it to fit the structure. That includes transforming the extracted data into a binary decision, of whether the checkbox is marked or not, for each checkbox. For that we will use a function:

[15]:
```
def is_marked(box):
    x, y, w, h, a = box
    # for now, checks if the area extracted is smaller than 80% of the␣
 ↪rectangle's area
    return a < (w * h) * 0.8
```

to test the function:

[16]:
```
for box in cboxes[0]:
    if is_marked(box):
        print("The box: ", box, " is marked")
```

```
The box:  [324 914  37  37 704]  is marked
The box:  [549 974  36  37 688]  is marked
The box:  [ 328 1367   37   37  745]  is marked
```

We also need to check whether the scanned exam matches the structure of the statement. That means it needs to have the same number of checkboxes, and they need to be positioned in a similar structure. For this we will use a function that compares the coordinates of the checkboxes on an exercise. The x and y coordinates differ in a matter of scale: the (w, h) of the boxes in the coordinates form pyPDF2 are around (13, 13), while in the coordinates from OpenCV they are around (37, 37). Also, the y coordinates are inverted in pyPDF2 when compared to in OpenCV, so it is better to compare distances than coordinates directly. So, we compare the distances from the first checkbox to the rest of the checkboxes, modifying them for scale.

[17]:
```
def is_same_exercise(st_checkboxes, scan_checkboxes):       # assumes␣
 ↪st_checkboxes and scan_checkboxes of same length
    st_init_x = st_checkboxes[0]['cords'][0]
    st_init_y = st_checkboxes[0]['cords'][1]
    st_init_w = st_checkboxes[0]['cords'][2]
    scan_init_x = scan_checkboxes[0][0]
```

```
    scan_init_y = scan_checkboxes[0][1]
    scan_init_w = scan_checkboxes[0][2]
    factor = scan_init_w / st_init_w                          # different scale␣
↪between the coordinates
    is_same = True
    for i in range(1, st_checkboxes.__len__()):
        st_x = st_checkboxes[i]['cords'][0]
        st_y = st_checkboxes[i]['cords'][1]
        st_dx_t = abs(st_x - st_init_x) * factor
        st_dy_t = abs(st_y - st_init_y) * factor
        scan_x = scan_checkboxes[i][0]
        scan_y = scan_checkboxes[i][1]
        scan_dx_t = abs(scan_x - scan_init_x)
        scan_dy_t = abs(scan_y - scan_init_y)
        # the difference in distance gets bigger as the distance grows, so␣
↪increase the range proportionally to distance
        # also, to avoid issues with alignment, increase by flat amount␣
↪relative to box size
        d_range = (scan_dx_t + scan_dy_t) * 0.2 + scan_init_w
        if st_dx_t < scan_dx_t - d_range or st_dx_t > scan_dx_t + d_range or␣
↪st_dy_t < scan_dy_t - d_range or st_dy_t > scan_dy_t + d_range:
            is_same = False
            # print('st_dx_t: ', st_dx_t, 'scan_dx_t: ', scan_dx_t, 'st_dy_t:␣
↪', st_dy_t, 'scan_dy_t: ', scan_dy_t)
    return is_same
```

test for the is_same_exercise function:

[18]:
```
# s_statement['exercises']
test_scan_boxes = [[ 324,  914,   37,   37,  704],
                   [ 399,  915,   37,   37, 1325],
                   [ 331, 1202,   37,   36, 1309],
                   [ 330, 1257,   37,   37, 1340]]
count_b1 = 0
count_b2 = 0
for exercise in s_statement['exercises']:
    count_b2 = count_b2 + exercise['checkboxes'].__len__()
    if not is_same_exercise(exercise['checkboxes'], test_scan_boxes[count_b1:
↪count_b2]):
        print("Error, estructura de casillas no encaja")
    count_b1 = count_b2
```

analyse function: takes the scanned pdf, finds the checkboxes, checks that the exams fits the statement, and exports a json file with the info on the marked boxes

[19]:
```
def analyse(scan_file, st_json, id_student):
    # get the checkbox data
```

```
    scan_boxes = find_cboxes(scan_file, id_student)
    # open and read the statement JSON
    with open(st_json) as json_file:
        data = json.load(json_file)
    # check if the same exam: check number of boxes
    if data['num_boxes'] != scan_boxes[0].__len__():
        print(data['num_boxes'], scan_boxes[0].__len__())
        print("Error, numero de casillas distinto")
    else:
        # modify st_json with the data from the checkboxes
        data['student'] = id_student
        # to group the boxes from the scanned pdf into exercises
        count_boxes1 = 0
        count_boxes2 = 0
        # iterate through the exercises in the statement dict
        for i, exercise in enumerate(data['exercises']):
            count_boxes2 = count_boxes2 + exercise['checkboxes'].__len__()
            # check if the same exam: check structure of boxes within exercises
            if not is_same_exercise(exercise['checkboxes'],␣
↪scan_boxes[0][count_boxes1:count_boxes2]):
                print("Error, estructura de casillas no encaja")
            # check each box in the exercise for if they're marked
            data['exercises'][i]['scan_marked'] = []
            for j in range(count_boxes1, count_boxes2):
                if is_marked(scan_boxes[0][j]):
                    data['exercises'][i]['scan_marked'].append(str(i) + ',' +␣
↪str(j-count_boxes1))
            count_boxes1 = count_boxes2

    # write the JSON file
    jf_name = 'json/scan_' + id_student+ '.json'
    os.makedirs(os.path.dirname(jf_name), exist_ok=True)
    with open(jf_name, 'w') as outfile:
        json.dump(data, outfile, indent=4)
```

example execution of analyse scan

[20]:
```
analyse('pdf/scanned_11111.pdf', 'json/st_Test.json', '11111')
```

4 Match

for now, just compares the marked checkboxes on both solution and scanned jsons

[21]:
```
def match(sol_json, scan_json):
    # open solution json
    with open(sol_json) as json_file:
        data_sol = json.load(json_file)
```

```
    # open scan json
    with open(scan_json) as json_file:
        data_scan = json.load(json_file)
    # check if same exam
    if data_scan['exam'] != data_sol['exam'] or data_scan['num_boxes'] !=␣
↪data_sol['num_boxes'] or data_scan['exercises'].__len__() !=␣
↪data_sol['exercises'].__len__():
        print('Error, solution y scanned no coinciden')
    count = 0
    for i, exercise in enumerate(data_sol['exercises']):
        if exercise['sol_marked'] == data_scan['exercises'][i]['scan_marked']:
            count = count + 1

    # print('Grade: ', 10*count/data_scan['exercises'].__len__())
    return {'num_correct': count, 'total_exercises': data_scan['exercises'].
↪__len__()}
```

example execution of match

[22]:
```
match('json/sol_11111.json', 'json/scan_11111.json')
```

[22]:
```
{'num_correct': 1, 'total_exercises': 2}
```

5 Global test

test with just one exam and student

[23]:
```
detect_st("pdf/statement_11111.pdf", "Test")
detect_sol("pdf/solution_11111.pdf", "json/st_Test.json", '1')
analyse("pdf/scanned_11111.pdf", "json/st_Test.json", '2')
match("json/sol_11111.json", "json/scan_11111.json")
```

[23]:
```
{'num_correct': 1, 'total_exercises': 2}
```

test with all exams, reading the student list from students.csv

[24]:
```
#set up the csv reader to get the id of each student
st_reader = csv.reader(open("students.csv"))
header = next(st_reader)

exam = "TestGlobal"
st_file = "pdf/st/question_"
sol_file = "pdf/sol/solution_"
scan_file = "pdf/scan/scanned_"

st_json = "json/st_" + exam + ".json"
sol_json = "json/sol_"
```

```
scan_json = "json/scan_"

student = next(st_reader)[2]
detect_st(st_file + student + ".pdf", exam)
detect_sol(sol_file + student + ".pdf", st_json, student)
analyse(scan_file + student + ".pdf", st_json, student)

grades = [{'student': student, 'grade': {}}]
grades[0]['grade'] = match(sol_json + student + ".json", scan_json + student + ␣
 ↪".json")

for row in st_reader:
    student = row[2]
    detect_sol(sol_file + student + ".pdf", st_json, student)
    analyse(scan_file + student + ".pdf", st_json, student)
    grades.append({
        'student': student,
        'grade': match(sol_json + student + ".json", scan_json + student + ".
 ↪json")
    })

print(grades)
```

```
[{'student': '12345', 'grade': {'num_correct': 0, 'total_exercises': 2}},
{'student': '31416', 'grade': {'num_correct': 1, 'total_exercises': 2}},
{'student': '271828', 'grade': {'num_correct': 2, 'total_exercises': 2}},
{'student': '11111', 'grade': {'num_correct': 1, 'total_exercises': 2}},
{'student': '22222', 'grade': {'num_correct': 0, 'total_exercises': 2}},
{'student': '33333', 'grade': {'num_correct': 1, 'total_exercises': 2}},
{'student': '44444', 'grade': {'num_correct': 2, 'total_exercises': 2}},
{'student': '55555', 'grade': {'num_correct': 2, 'total_exercises': 2}},
{'student': '666666', 'grade': {'num_correct': 2, 'total_exercises': 2}}]
```

[24]: