

bitacora

Andres Bartolome

January 2024

# Chapter 1

## Bitácora

- 11OCT22: Reu en Navales. jviu, pang, victor, carlos:
  - aprendemos a usar latex
  - Hablamos sobre la web git de pyexams [1]
  - Miramos textsurgery, sus test unitarios, etc
  - Carlos recomienda que hablemos del propósito de pyexams.
  - Instala textsurgery, corre todos los tests, tb los de la doc., documenta los problemas que surjan...
  - Pedimos permiso para overleaf para carlos
- Semana 11OCT-18OCT
  - Instalado textsurgery y ejecutado pruebas unitarias
  - Instalado pyexams y ejecutado ejemplos python/en y /python/es, 'es' funciona sin error, 'en' falla
- 18OCT22: Reu en Navales. jviu, pang, victor:
  - Trabajo simultáneo con overleaf
  - Juan manda deberes para aprender LaTeX: crear un examen
  - Comprobamos que tras instalar pyexams algún ejemplo falla por no tener instalados todos los paquetes texlive
- Semanas 18OCT-02NOV
  - TODO: Documentar estado de textsurgery y pyexams, primeras impresiones, en capítulo estadocuestion
  - TODO: Aprender latex con el manual de wikibooks
  - TODO: examen de prueba

1. Me he basado en los ejemplos de exámenes de pyexams, para generar el examen de ejemplo proporcionado.
  2. Escrito la versión 1 del examen en LaTeX.
  3. Transformado el documento para que la ejecución con pyexams genere las dos versiones del examen
  4. Ejecutado con pyexams con los siguientes resultados
    - (a) Error por encoding, el fichero utilizado tenia un encoding distinto a UTF-8, cambiado el encoding a UTF-8 con el editor de texto.
    - (b) Error por formato de las sentencias *sif*, corregido.
    - (c) Ejecutado sin errores en la terminal, pero no he declarado bien las partes de ejercicio y de solución para pyexams (sale tanto el ejercicio como la solución en ambos).
    - (d) Declarado la pregunta y la solución dentro del fichero *tex*, ahora el enunciado se genera correctamente pero la solución no. El pdf de solución tiene la primera página en blanco (solo con la cabecera) y la solución se corta en la segunda pagina.
- TODO: Averiguar/Documentar los paquetes *texlive* que hay que instalar para que los ejemplos de pyexams funcionen.
  - TODO: <https://framagit.org/pang/pyexams/-/issues/5>  
 Añadir un argumento extra para el archivo *students.csv* en el envío de correos.
    - \* Primero he querido probar el funcionamiento del envío de correos, utilizando las plantillas de ejemplo de pyexams.
      - Al poner el envío con mi correo de la universidad me daba un problema del servidor SSL, probablemente he puesto el puerto mal. He utilizado un correo de gmail para el envío, y he tenido que crear una contraseña de aplicación por la configuración de seguridad de google.
      - Ha dado un error al no existir el fichero pdf que enviar. La opción *send* necesita que generes primero el examen.
    - \* Los argumentos que recibe pyexams se tratan con la librería python *argparse*, así que he mirado la documentación de esta librería y los otros argumentos de pyexams para crear el nuevo argumento
      - Añadido un nuevo argumento (*-sf fichero.csv*) que recibe el nombre del fichero, con valor por defecto *students.csv*.
      - Probado el funcionamiento de tanto el nuevo argumento como el envío normal por defecto.
  - TODO: Resuelve <https://framagit.org/pang/pyexams/-/issues/2>
    1. Mira en *texsurgery* cómo se copia *texsurgery.sty* a la carpeta de trabajo: <https://framagit.org/pang/texsurgery/-/commit/6d4e05950f953a1ff8932a058c5421b6dd8924bb>

- \* En *setup.py*: *include\_package\_data=True* e incluye el paquete con los datos
    - \* En *command\_line.py*: comprueba el argumento y copia el fichero con *shutil.copy()*. Obtiene la ubicación del fichero a partir de *texsurgery.\_\_path\_\_[0]*.
  - 2. Mira en [https://framagit.org/pang/pyexams/-/blob/master/pyexams/command\\_line.py](https://framagit.org/pang/pyexams/-/blob/master/pyexams/command_line.py) cómo se añaden opciones a la línea de comandos
  - 3. Prepara una plantilla para un examen básico, o para un envío de emails básico
    - \* plantilla de examen en latex
    - \* plantilla de configuración de correo (hay una en examples)
    - \* plantilla de correo a enviar (hay una en examples)
  - 4. une la línea de puntos
  - 5. piensa cómo hacerlo con plantillas por lo menos en castellano e inglés
- 02NOV22: Reu en Navales. pang:
    - Arreglado el error por el que generaba la solución en una sola página, con la solución de Auto Multiple Choice. Sustituye el entorno *question* por un nuevo comando. Este comando comprueba una variable de AMC e imprime la solución cuando debe.
    - Issue 5:
      1. Cambiado el argumento para almacenarlo en formato *string* en vez de *List*.
      2. Configurado la clave SSH para framagit.
      3. Subido la solución a una nueva rama del repositorio: *issue5*
  - Semana 02NOV-08NOV
    - Cerrar el issue 5
    - Continuar el trabajo
  - 08NOV22: Reu en Navales. pang:
    - TODO: crear issue con la opción de generar a LateX, en vez de generar pdf. Basta con no borrar el tex que ya generamos (cuando hay error no se borra, de hecho).
      - \* Creado el Issue 7
    - Pensamos si pandoc puede ayudar a convertir a jupyter ipynb, pero también dudamos si este issue puede interesar al público en general.

- Pensar: escanear exámenes. Puede ser útil <https://stackoverflow.com/questions/22898145/how-to-extract-text-and-text-coordinates-from-a-pdf-file> (pang conocía <https://pypi.org/project/PyPDF2/> pero no parece que sirva para sacar las coordenadas del formulario, sólo los campos). también puede ayudar mirar el código de R-exams: <https://cran.r-project.org/web/packages/exams/index.html>. En la siguiente reunión (22NOV), pang intentará detallar los pasos y/o invitar a más gente. Después de esa reu, deberíamos quedar con Sandra.
- Semanas 08NOV-22NOV
  - Escanear exámenes
    - \* Id de alumno: cuadros del encabezado
    - \* Lectura del examen: varias páginas, formato (pdf, jpg, ...)
    - \* Para cada ejercicio:
      - # de ejercicio dentro del examen
      - identificar los cuadros y sus posiciones (OpenCV 1, 2, 3, 4; paquete boxdetect)
      - cuadros marcados
    - \* obtener la solución
      - según el id del alumno
      - a partir del pdf con la solución (detectando las cajas como con el escaneado, pero posiciones distintas ya q puede tener explicación)
      - a partir de otro archivo q generar: pdf solución simplificado (que tenga las mismas posiciones que el enunciado, simplificaría el escaneo), o txt con datos sencillos (p.ej: # de alumno, # de ejercicios, # numero de cuadros, # cuadro solucion)
- 22NOV22: Reu en Navales. pang, :
  - TODO: Estudiar como obtener la informacion de pdfminer: casillas, tags y coordenadas
  - TODO: definir modulos para escanear examenes, inputs/outputs
- 25NOV22: Reu en Teams. pang, sandra:
  - Definir distintos tipos de preguntas que acepta pyexams
  - Diagrama de componentes/clases
  - Lista de requisitos -¿ Hacer una entrevista de requisitos en navales
  - Metodologias Agiles -¿ sacar de la Bitacora y la lista de requisitos los sprints

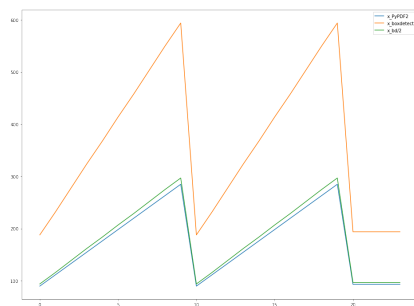


Figure 1.1: Comparación de las coordenadas x

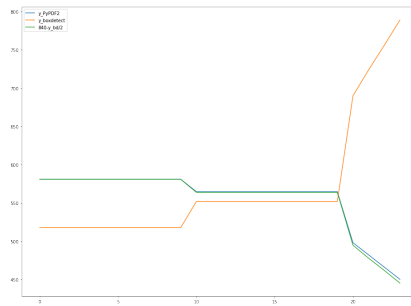


Figure 1.2: Comparación de las coordenadas y

- Definir los conocimientos utilizados para la realización del TFG y de donde se ha obtenido
- Documentar mejor las pruebas realizadas con imágenes
- Utilización de cuadernos jupyter para la realización de pruebas
- Semana 22NOV-29NOV
  - Creado un diagrama inicial de los módulos de Escanear Exámenes (en IdeasSeltas)
  - Probado en un cuaderno jupyter tanto los visitors de pyPDF2 como la librería boxdetect
- 29NOV22: Reu en Navales. pang:
  - Cambiando pyPDF2 en constantes - FieldDict Attributes Rect consigue los rectángulos (correo).
  - TODO comparar las coordenadas de pyPDF2 y boxdetect
- Semanas 29NOV-13DIC
  - Realizado los cambios de pang a PyPDF2 en local
  - Comparado las coordenadas de las casillas en PyPDF2 y boxdetect. Las coordenadas parecen estar escaladas 1:2, y la coordenada y está invertida
  - Generado exámenes escaneados para probar la detección de casillas (Ideas Seltas)
  -
- 13DIC22: Reu en Navales:
  - En mi ordenador no genera las casillas binarias con identificadores correctos, estudiar posibilidad de utilizar códigos qr (pang) con la información (examen, copia, número de página) en formato JSON

- Uso de los puntos de las esquinas para corregir la perspectiva (perspective correction)
- Definido 'escanear exámenes' en 3 funciones:
  - \* Detect: 'detectar diseño', obtiene de los pdfs solución generados las coordenadas de cada casilla, el estado correcto y los datos del alumno; que extrae a un fichero json
  - \* Analyse: en el fichero pdf con los exámenes escaneados, identificar cada examen, página, ejercicio y casilla a partir de los identificadores (puntos de las esquinas, casillas binarias, código qr) y las coordenadas de las casillas; extraer también a json
  - \* Match: a partir de las salidas de las otras dos funciones, puntuar cada ejercicio
- División del problema en capas: Grupo  $\hookrightarrow$  Examen  $\hookrightarrow$  Papel  $\hookrightarrow$  Pregunta  $\hookrightarrow$  Casilla
- Diagramas de tareas/etapas/flujo (posibilidad de omitir etapas según los flujos: grade en caso de encuestas)
- Semana 13DIC-20DIC
- Diseño de prototipo de función "Escanear exámenes", TODO:
  - TODO: Detectar diseño
    - \* Leer alumnos.csv y de ahí obtener cada fichero pdf de solución
    - \* Leer de (el fichero tex / los pdf de solución) el número y nombre de ejercicios y el número de casillas
    - \* Obtener de los pdf de solución las casillas, sus coordenadas y estado (on/off)
    - \* Exportar los datos a JSON
  - TODO: Analizar escaneo
    - \* Leer el pdf con los exámenes escaneados página a página
    - \* Obtener la información de cada página (examen, copia y número página)
    - \* Identificar las casillas y sus estados
    - \* Exportar los datos a JSON
  - TODO: Match
    - \* Obtener la información de los JSON de detect y analyse
    - \* Puntuar cada examen
- Diseño de prototipo de función "Escanear exámenes", pruebas realizadas:
  - "t\_json.ipynb": pruebas del formato de datos a guardar en JSON y de la lectura y escritura de estos datos.

- "t\_pdftopng.ipynb": ya que para identificar las casillas con cv2 se trabaja con ficheros png, hay q transformar los ficheros pdf en 1/varios pngs. Para esta transformación pruebo con la librería pdf2image. También contiene una prueba de identificar casillas de pngs mediante cv2.
  - "t\_scan.ipynb": prototipo de la función "Escanear exámenes", con funciones *detect*, *analyse* y *match*.
  - "checkboxes.py": funciones utilizadas en el prototipo: *find\_checkboxes*, *check\_marcked* y *match*.
- 20DIC22: Reu en Navales:
    - Dividir por capas la función "Escanear exámenes":
      - \* Examen: considerando un mismo examen generado con pyexams a partir de un fichero tex
      - \* Variante: para cada una de las variantes generadas con pyexams, con distintas soluciones
      - \* Alumno\*: para los distintos estudiantes que realicen una misma variante (\*¿Como identificar el examen de cada alumno?)
      - \* Ejercicios: identificar los ejercicios y sus casillas, para detectar el patrón del examen y la solución, utilizar dicho patrón para detectar las casillas en el examen escaneado y comprobar con la solución.
      - \* Casillas: detectar las casillas, sus coordenadas y si están marcadas
    - Añadir QR a examenes desde pyexams:
      - \* Librería qrcode de latex
      - \* Añadir funcionalidad a pyexams para insertar un codigo qr, haciendo uso de la libreria qrcode, en el lugar indicado con '`\QR`': *'replace(\QR, JSON)'*
  - 20DIC22-20ENE23
    - Definición de la función 'Escanear examen'
      - \* Dividido la función en 4 funciones que actúan en distintas capas:
        - \* Detect Statement: obtiene de un fichero pdf generado por pyexams el diseño del examen: sus ejercicios y las casillas de cada ejercicio, junto a las coordenadas de las casillas. Esta función actúa en la capa de examen, ya que en teoría las variantes de los exámenes no modifican la posición de las casillas (\*las variantes podrían tener distintas coordenadas, pero no debería ser un problema). Guarda los datos obtenidos en un fichero JSON



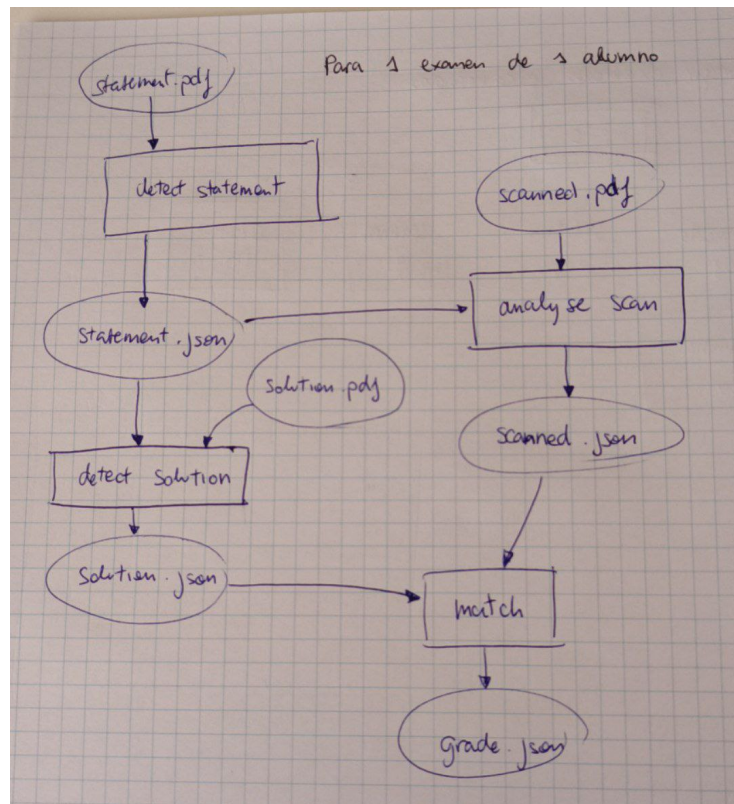


Figure 1.3: Diagrama de componentes de la función Escanear Exámenes

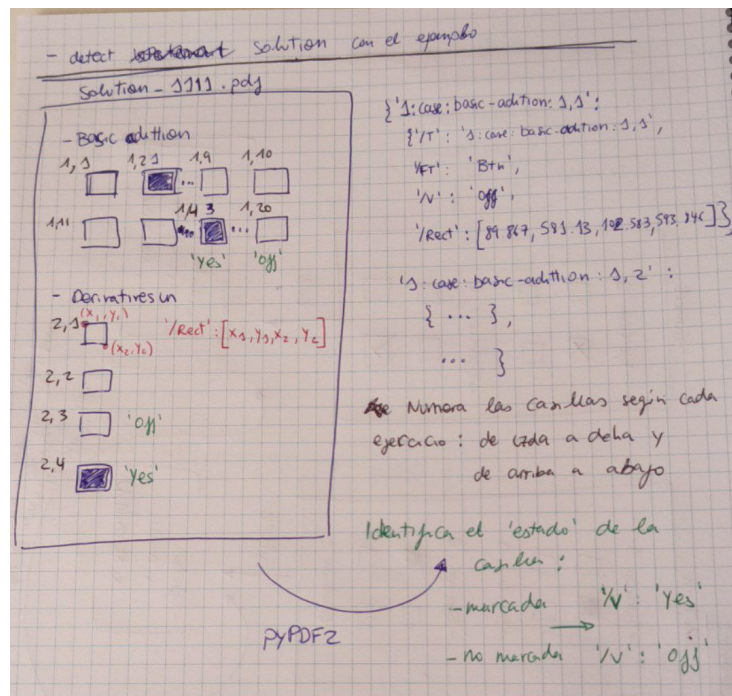


Figure 1.4: Ejemplo de Detect Solution

- \* Detect Solution: obtiene de un fichero pdf generado por pyexams las casillas marcadas en cada ejercicio en la solución del examen. Esta función actúa en la capa de variante, ya que la solución de cada variante es diferente. Guarda los datos en un fichero JSON que genera a partir del generado en detect statement.
  - \* Analyse Scan: obtiene de un fichero pdf con el examen 'escaneado' realizado por un alumno: las casillas de cada ejercicio y si están marcadas. Para detectar las casillas hacen falta obtener imágenes de cada pagina, por lo que primero genera estas. Detecta las casillas y sus coordenadas con OpenCV, además de su estado (si están marcadas o no). Compara las casillas leídas con la estructura obtenida en detect statement, y guarda la información de qué casillas hay marcadas. Funciona en la capa de alumno. Guarda la información de las casillas marcadas en un JSON modificando el generado en detect statement
  - \* Match: Compara los ficheros JSON generados en detect solution y analyse scan para valorar la puntuación del examen.
- Prototipo de Escanear exámenes
- \* Esquema de funcionamiento: figura 1.6

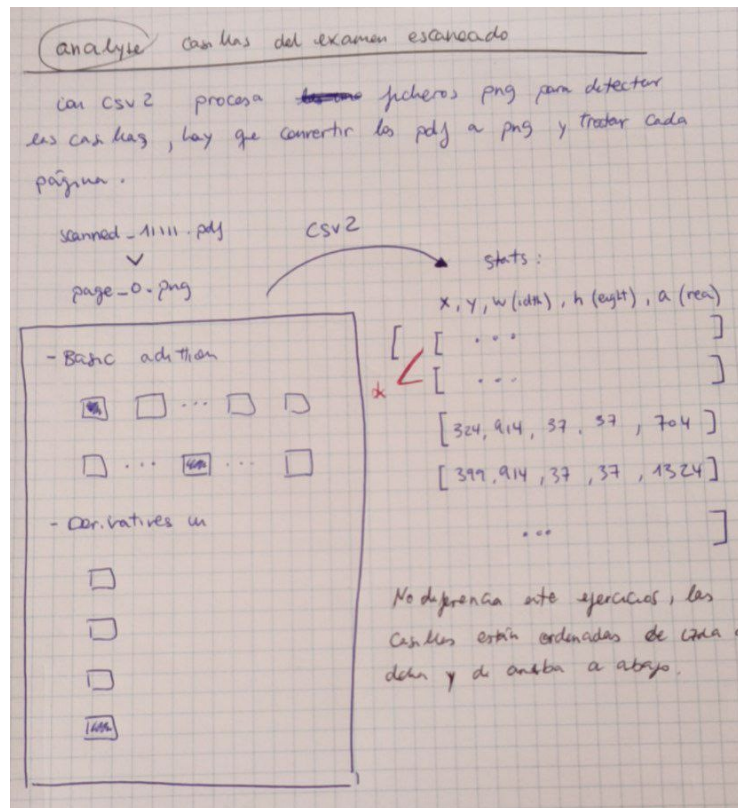


Figure 1.5: Ejemplo de Analyse Scan

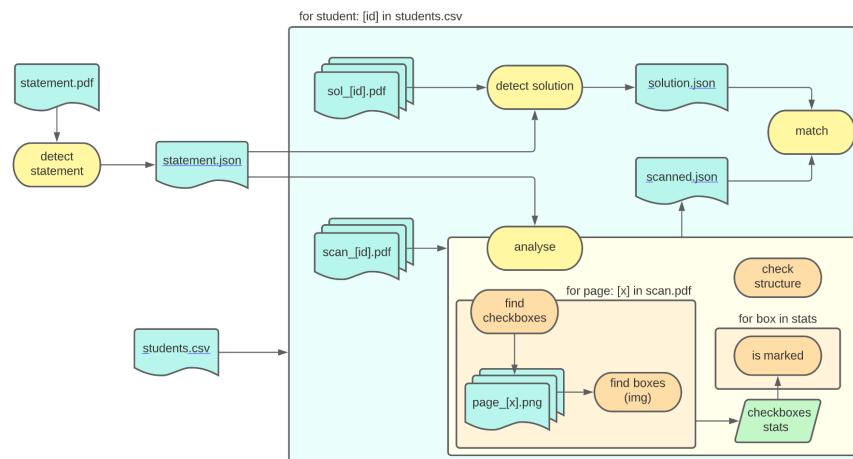


Figure 1.6: Esquema del prototipo de Escanear exámenes

- Detect statement: lee un único enunciado del examen y genera un json con la estructura de este: ejercicios, casillas y coordenadas de estas. Utiliza la versión modificada de pyPDF2 para detectar las casillas.
- Para cada alumno en students.csv:
  1. Detect solution: lee la solución de la variante, comprueba que la estructura es igual a la del enunciado y guarda tanto la estructura como las casillas que están marcadas en un json. Utiliza la versión modificada de pyPDF2 para detectar las casillas.
  2. Analyse: lee el examen rellenado por el alumno, comprueba que la estructura es igual a la del enunciado y guarda tanto la estructura como las casillas que están marcadas en un json. Utiliza OpenCV para detectar las casillas, en las siguientes funciones:
    - (a) Find checkboxes: la forma de detectar las casillas con OpenCV utiliza de entrada un fichero png, así que esta función convierte cada pagina del pdf a un png y detecta las casillas en cada pagina.
    - (b) Is marked: la información que extrae OpenCV sobre las casillas, que se puede ver en la figura 1.7, incluye información de sus coordenadas (x e y), tamaño (ancho y altura) y área interior a la casilla. Este área parece ser mucho menor en las casillas marcadas, así que la función compara este parámetro con el área del rectángulo (an-

cho por altura) y considera que la casilla está marcada si el parámetro es menor al 80% del área del rectángulo.

3. Match: compara los json de solucion y escaneado y devuelve el numero de ejercicios 'bien' (con las mismas casillas marcadas) y el total de ejercicios

\* Cuaderno ipython del prototipo:

# 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,
                                  12.7159999999999894
                              ]
                          },
                          {
                              "checkbox": "0,1",
                              "cords": [
                                  111.549,
                                  563.517,
                                  12.715999999999994,
                                  12.7159999999999894
                              ]
                          }
                      ]
                  },
                  "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-addition',
        '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-addition",
                    "checkboxes": [
                        {
                            "checkbox": "0,0",
                            "cords": [
                                89.867,
                                563.517,
                                12.715999999999994,
                                12.7159999999999894
                            ]
                        },
                        {
                            "checkbox": "0,1",
                            "cords": [
                                111.549,
                                563.517,
```

```

        12.715999999999994,
        12.715999999999894
    ],
    },
    "scan_marked": ["0,1"] },
    ↪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 from 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': '66666', 'grade': {'num_correct': 2, 'total_exercises': 2}}]

```

[24]:

```
1 cboxes = find_cboxes('pdf/scanned_11111.pdf', '11111')
2 cboxes

▼ [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],
```

Figure 1.7: Información de las casillas obtenida con OpenCV

- 20ENE23: Reu en Teams:
  - TODO: Añadir la funcionalidad de escanear exámenes a pyexams:
    - \* Generar los json de statement/solution al compilar el tex
    - \* Funcion scan: "pyexams -scan ruta [fich.json]"
      - ruta: si es un pdf, analizar cada examen que se encuentre en el pdf. Si es un directorio, analizar cada pdf dentro de este
      - fich.json: identifica cada alumno con una variante de examen?
  - TODO: funcionalidad QR: añadir comando \QR que pyexams reemplace por un código QR (libreria qrcode de latex) con datos del examen: [numero de pagina,] variante, id del examen, etc.
  - TODO: hablar con los de pyPDF2 para incluir los cambios. También: instrucciones de instalación con pyPDF2 modificado
  - TODO: Analyse alternativo para pdfs rellenos con editor/no escaneados. Realizar primero una búsqueda de casillas con pyPDF2, y si no encuentra las casillas buscar con OpenCV.
  - TEST: pruebas mas extensas de escanear exámenes: examen de más páginas, función shuffle, etc..
  - TEST: probar distintas formas de rellenar casillas: opaca, cruz, tick, etc..
  - TODO: guardar la información de página en analyse (¿cómo identifica distintas paginas pyPDF?)

- TODO: Transformar las paginas escaneadas antes de analizar con OpenCV para rectificar (transformación del qr, puntos de las esquinas de AMC)
  - TODO: guardar las coordenadas [de solucion y] de escaneado en el json
  - TODO: funcion match, informacion a guardar en el JSON: para cada ejercicio guardar id, casillas en solucion y en escaneado, y si esta bien
  - [TODO: guardar en .pyexams información de los proyectos ejecutados en el PC]
- 24ENE23: Reu en Navales
    - connected with stats de cv2 parece no detectar cuadrados enteros como casillas
    - sacar los dpi y min\_width de las funciones de escanear examenes para que sean variables, y obtenerlos en funcion del pdf
    - trabajo futuro descrito en el correo de pablo tras reunion teams
    - generar un qr en cada cara, si se usa para corregir ángulo
    - click derecho en un archivo para corregir
  - 24ENE23-x
    1. Pruebas con casillas rellenas:
      - para las pruebas uso un examen obtenido del perfil de github de mmoralesf
      - modificar el examen para generar con pyexams  
Al crear las preguntas por separado como questions, la solucion no generaba bien la opcion correcta (figura 1.8).  
Creado dentro de elementos e insertado como en el ejemplo de AMC con:  
`\cleargroup{BigGroupe}`  
`\copygroup{elemento}{BigGroupe}`  
`...`  
`\retituegroupe{BigGroupe}`
      - generar al menos una versión (usar shuffle para las casillas)  
2 versiones distintas con alumnos.csv  
¿cómo activo el shuffle? \*
      - ¿distintas formas de rellenar casillas?
      - rellenar las casillas de distintas formas y corregir
    2. Shuffle
      - La opción shufflechoices está desactivada por defecto, al activar produce un error al faltar el parámetro 'childrenselector' en la llamada a TexSurgery.shuffle(), como se puede observar en la figura 1.9

CORRECTED

Primavera 2018  
Examen Extraordinario de Matemáticas II  
2do Semestre  
Salón 41

- Question 1** Si  $a$  es una solución para una ecuación, entonces  $(x+a)$  es un factor de la ecuación.
- ☐ Falso  
☐ Verdadero
- Question 2** El coeficiente de  $5x^2$  es 2.
- ☐ Falso  
☐ Verdadero
- Question 3** Los ángulos son funciones de los lados
- ☐ Falso  
☐ Verdadero
- Question 4** Todo triángulo acutángulo tiene sus tres lados de distinta longitud
- ☐ Falso  
☐ Verdadero
- Question 5** La suma de los ángulos interiores de un hexágono es de  $720^0$
- ☐ Falso  
☐ Verdadero

Figure 1.8: Solucion generada mal: casillas correctas no aparecen

```
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx:~/Downloads/test_casillas_marcadas$ pyexams examen_mult.tex -both -all
joblib.externals.loky.process_executor._RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/externals/loky/process_executor.py", line 428, in _process_worker
    r = call_item()
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/externals/loky/process_executor.py", line 275, in __call__
    return self.fn(*self.args, **self.kwargs)
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/_parallel_backends.py", line 620, in __call__
    return self.func(*args, **kwargs)
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/parallel.py", line 288, in __call__
    return [func(*args, **kwargs)
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/parallel.py", line 288, in <listcomp>
    return [func(*args, **kwargs)
  File "/home/andres/Documents/pyexams/pyexams/pyexams.py", line 221, in one_pdf
    ts.shuffle('choices')
TypeError: TextSurgey.shuffle() missing 1 required positional argument: 'childrenselector'
"""

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/andres/.local/bin/pyexams", line 33, in <module>
    sys.exit(load_entry_point('pyexams', 'console_scripts', 'pyexams')())
  File "/home/andres/Documents/pyexams/pyexams/command_line.py", line 116, in main
    do_process(opts)
  File "/home/andres/Documents/pyexams/pyexams/command_line.py", line 107, in do_process
    ex.to_pdf(opts.with statement, opts.with solution, opts.all, shufflechoices=True)
  File "/home/andres/Documents/pyexams/pyexams/pyexams.py", line 144, in to_pdf
    Parallel(n_jobs=8)(
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/parallel.py", line 1098, in __call__
    self.retrieve()
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/parallel.py", line 975, in retrieve
    self._output.extend(job.get(timeout=self.timeout))
  File "/home/andres/.local/lib/python3.10/site-packages/joblib/_parallel_backends.py", line 567, in wrap_future_result
    return future.result(timeout=timeout)
  File "/usr/lib/python3.10/concurrent/futures/_base.py", line 458, in result
    return self._get_result()
  File "/usr/lib/python3.10/concurrent/futures/_base.py", line 403, in __get_result
    raise self._exception
TypeError: TextSurgey.shuffle() missing 1 required positional argument: 'childrenselector'
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx:~/Downloads/test_casillas_marcadas$
```

Figure 1.9: Error al activar la opción de shuffle



**Question 1** Si  $a$  es una solución para una ecuación, entonces  $(x+a)$  es un factor de la ecuación.

- ☒ Falso  
☐ Verdadero

**Question 2** El coeficiente de  $5x^2$  es 2.

- ☒ Falso  
☐ Verdadero

**Question 3** Los ángulos son funciones de los lados

- ☒ Falso  
☐ Verdadero

**Question 4** Todo triángulo acutángulo tiene sus tres lados de distinta longitud

- ☒ Falso  
☐ Verdadero

**Question 5** La suma de los ángulos interiores de un hexágono es de  $720^\circ$

- ☐ Falso  
☒ Verdadero

**Question 6** El semiperímetro de un triángulo equilátero con base  $b = 5$  cm es  $s = 7,5$  cm.

- ☒ Verdadero  
☐ Falso

examen,variante,1:



Figure 1.10: Generación del código QR y texto en claro

### 3. Añadir función QR:

- nueva branch en pyexams
- guardar info de examen, variante y pagina ¿cómo obtener el número de pagina? el comando `\thepage`
- la posición debería ser fija si se va a utilizar en reconocimiento de posición/inclinación ¿en el pie de pagina?  
pie de pagina: probado librería *fancyhdr*, al meter en un fichero  $\text{\LaTeX}$  de pyexams no lo genera, ¿AMC se lo carga?  
marca de agua: librerías *xwatermark*, no consigo que compile, y *draftwatermark*, genera el qr correctamente en cada pagina
- probar en latex el código para generar qr's, figura 1.10
- añadir a pyexams una función que cambie los comandos `\QR` por el código probado
- añadido a la rama "scan" de pyexams en el commit "Added function to insert qr codes with the exam id, variant id and page":
  - \* Función `qr(tex.in, exam_id, variant_id, watermark_params)`:  
Parámetros:  
`tex.in`: texto del fichero `tex` tras procesar con `code_surgery`

exam\_id: nombre del fichero tex a procesar, identificador del examen

variant\_id: id de estudiante, de students.csv, identificador de variante de examen

watermark\_params: parámetros para generar la marca de agua (escala, ángulo de giro, color, posición base, y tamaño), con valores por defecto.

Ejecución:

Genera un string con el código latex que genera un QR con exam\_id, variant\_id y el número de página como una marca de agua con las características indicadas en watermark\_params.

Si encuentra el código “\\QR” en tex\_in, lo reemplaza por el string mencionado.

Devuelve en tex\_out el texto, modificado o no

- \* dentro de la función one\_pdf, que genera un pdf de enunciado/solución para un alumno, y tras procesar el fichero tex con code\_surgery, llama a la función qr pasándole el código tex, el nombre del fichero tex y el id del alumno.

#### 4. Añadir función de escanear exámenes, partes:

- DETECT: Generar json's al crear enunciados y soluciones
- ANALYSE: Añadir función -scan : analizar exámenes escaneados/rellenados
- MATCH: corregir los exámenes con los json ya generados

TODO:

- crear branch en pyexams
- DETECT: Añadir funciones detect\_statement y detect\_solution a pyexams: reciben el fichero enunciado/solución y crean el json correspondiente
- DETECT: Llamar a las funciones detect\_X después de crear los pdf's correspondientes dentro de pyexams
- dependencias de las nuevas funciones: PyPDF2 modificado, pdf2image, OpenCV (cv2 en python, existen 4 paquetes distintos en pip, uso opencv-python)
- ANALYSE: añadir función scan “pyexams -scan file”
- ANALYSE: intentar extraer las casillas con pyPDF2 (rellenado) antes de hacerlo con OpenCV (escaneado)
- ANALYSE: (escaneado) leer el QR  
no está detectando correctamente el QR, dependiendo del dpi del png detecta los puntos o no, pero no lo decodifica en ningún caso
- + mejorar la imagen antes de buscar el QR
- + ¿cómo funciona qr\_detect de OpenCV?

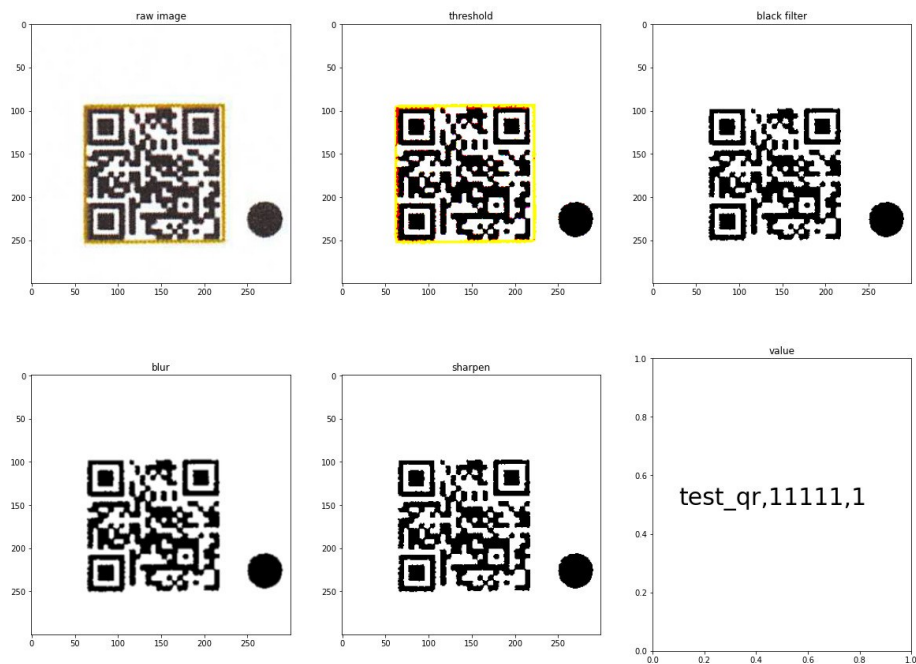


Figure 1.11: Procesamiento del QR paso a paso

Conseguido reconocer los 6 qrs de los exámenes de prueba, con el siguiente procesamiento:

Umbralizado la imagen con valor umbral 120

Aislado los píxeles negros de la imagen (eliminar un color específico)

Suavizado (Gaussian blur (3, 3)) y agudizado (sharpen ([-1,-1,-1],[-1,9,-1],[-1,-1,-1])) la imagen sucesivamente

Buscado y decodificado el código qr con QRCodeDetector de OpenCV

Detecta correctamente los qrs. Figura 1.11: procesamiento de la imagen para decodificar el QR paso a paso, figura 1.12: QRs recortados tras procesar

- ANALYSE: (escaneado) extraer las casillas  
 el tamaño de la imagen depende del dpi, parametrizar el tamaño mínimo de línea para detectar las casillas (dpi: 250 , min\_lin: 35 -> min\_lin: dpi \* 0.14)  
 la función de OpenCV detecta rectángulos con lados mayores al mínimo, hay que identificar cuales de estos rectángulos son casillas y descartar los que no lo son:
  - \* cuadros de texto: tendrán un tamaño mayor al de las casillas, descartar si uno de los lados es mayor al 150% del mínimo

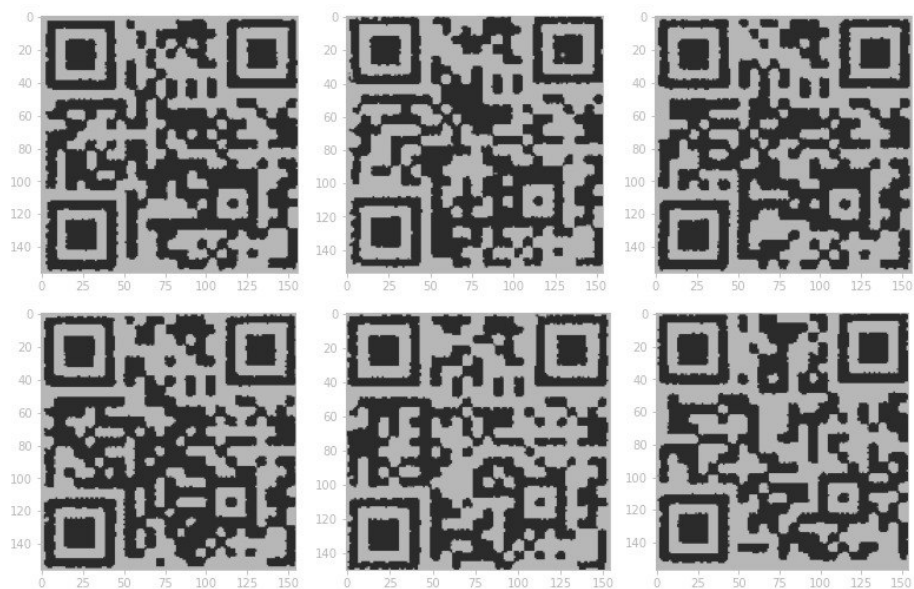


Figure 1.12: Códigos QR del examen de prueba

- \* cuadrados de AMC: están en el cabecero, descartar si la coordenada 'y' es menor al 10% del alto del documento (obtener el alto del documento a partir del dpi, o del min\_lin:  $\text{alto} = 11.6 * \text{dpi}, 82.8 * \text{min\_lin}$  )
- \* puntos del QR : sus coordenadas se corresponden a las coordenadas del QR, descartar si se encuentran en un rango basado en las coordenadas del QR

La función `connectedComponentsWithStats()` tiene problemas para reconocer casillas marcadas rellenas de forma manual, cambiado a la función `findContours()`, que encuentra los contornos de las casillas (ejemplo: 'square detection in image').

La función tiene problemas para detectar casillas con bordes muy finos, hay que hacer un pre-procesar la imagen: pasar blanco y negro, suavizar (blur), agudizar (sharpen) y umbralizar (threshold de Otsu)

- ANALYSE: (escaneado) identificar si las casillas están marcadas  
Extraer el interior de la casilla de sus coordenadas y la imagen binarizada

Ignorar los píxeles cercanos al borde

Si el valor del píxel es mayor a un umbral, sumar uno a un contador, para todos los píxeles

Si el contador es mayor al 10% del área del interior de la casilla considerar que está marcada

## 5. Corregir orientación

- Detectar los círculos de las esquinas (Auto Multiple Choice, J. M. M. S Kularathna)
- Hecho:
  - \* Añadido la función `\QR` a pyexams: detecta '`\QR`' en los ficheros  $\LaTeX$  y lo reemplaza por código que genera un QR con los datos de examen, variante y página en la esquina inferior izquierda de cada página del examen (commit 7a32d1f2)
  - \* Añadido la función que genera los ficheros JSON con la estructura de los ejercicios y casillas de tanto enunciados como soluciones en la función que genera pdf's de pyexams (commit caede94e)
  - \* Generado un fichero  $\LaTeX$  que genera un examen de tres páginas con veinte preguntas de opción múltiple
- 10FEB23: Reu en teams
  - editar imagen en GIMP para cambiar la perspectiva y probar el reconocimiento del qr.
- 10FEB-24FEB23
  - Mejorado el preprocesamiento de imagen en la detección de casillas. Inicialmente (figura 1.13), pasaba la imagen a blanco y negro y la umbralizaba. Tras los cambios (figura 1.14), la pasa a blanco y negro, suaviza, agudiza y ubraliza con umbral de Otsu. Esto ayuda a detectar las líneas que forman las casillas
  - Cambiado la función que detecta las formas de las casillas. Antes utilizaba `connectedComponentsWithStats`, que dividía casillas marcadas en varios componentes si el relleno era lo suficientemente grande. La función `findContours` encuentra los contornos externos, y no divide la casilla. Comparación en la figura 1.15.
  - Clasificación de los contornos detectados. La función detecta varios contornos (figura 1.16), de los cuales no todos son casillas. Además de casillas, detecta también los cuadros del cabecero de Auto Multiple Choice, cuadros de texto y formas dentro del QR. Descarto estos tipos de contornos según el tipo: cabecero AMC - posición 'y' inferior al 10% del tamaño de página, cuadros de texto - ancho o largo considerablemente distinto al tamaño de línea definido, y formas del QR - coordenadas interiores a las de los puntos del QR
  - Detección de casillas marcadas: extraído los interiores de las casillas a partir de las coordenadas obtenidas de cada contorno, descartado píxeles cercanos al borde para reducir ruido y contado el área rellena. Si este área supera un umbral del tamaño de la casilla, se considera que la casilla está marcada (Ejemplo en la figura 1.17).

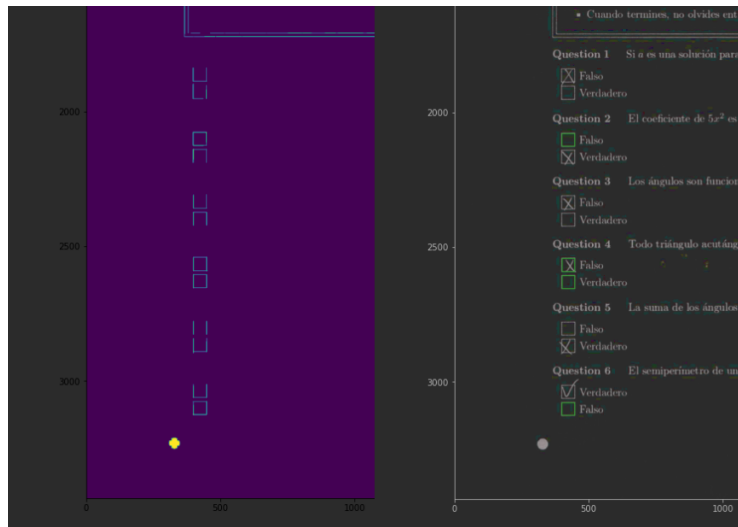


Figure 1.13: Preprocesamiento inicial

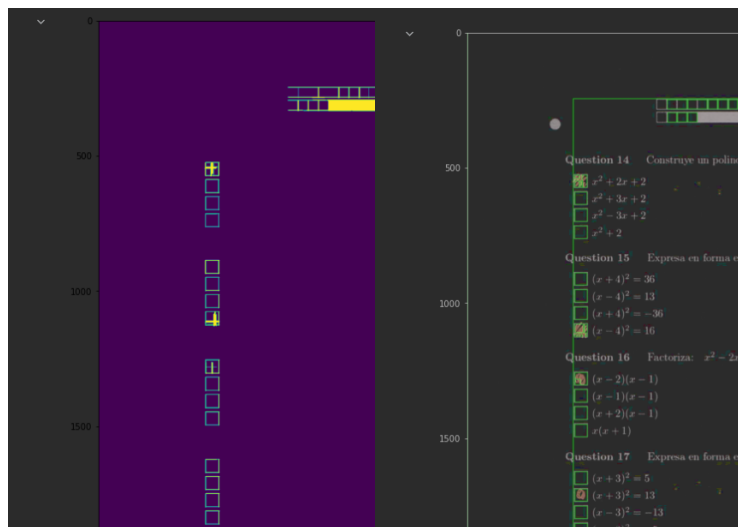


Figure 1.14: Preprocesamiento final

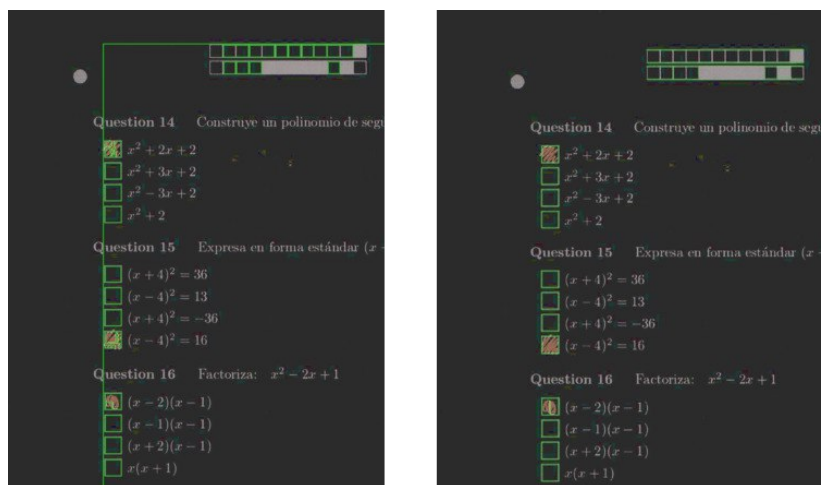


Figure 1.15: Componentes conectados - contornos

- Detección de códigos QR: separado de la detección de casillas y cambiado el preprocesamiento. En este, realiza primero el umbral, extrae los píxeles en negro de la imagen, y por último suaviza y agudiza repetidas veces hasta que encuentra el QR. Este método ha funcionado con los seis codigos de ejemplo, pero no para todas las calidades dpi, siendo poco fiable. (Ejemplo en la figura 1.18).

- 24FEB23: Reu en teams, pang

- Por correos antes de la reunión:

Resumen de los cambios realizados.

Definición de dos posibles usos de 'pyexams -scan':

- \* 'pyexams -scan file.pdf --exam-data DATOS.json'
- \* 'pyexams -scan file.pdf'

Leer del código QR el identificador de examen y variante.

Si el fichero JSON no se recibe de parámetro, buscar en la ubicación actual o en .pyexams/history o similar.

Leer el fichero JSON, con la información del examen: ejercicios, casillas y coordenadas

- Utilizar coordenadas de las casillas (obtenidas con pyPDF2 al generar el pdf, en JSON de enunciado) para identificar los contornos que son casillas. Mejor que descartar los contornos que no pueden ser casillas, reduce error.
- Parametrizar el valor umbral de detección de casillas marcadas, manteniendo el valor por defecto.
- Guardar el número de página dentro de la información de cada casilla (tanto en detectar como analizar)

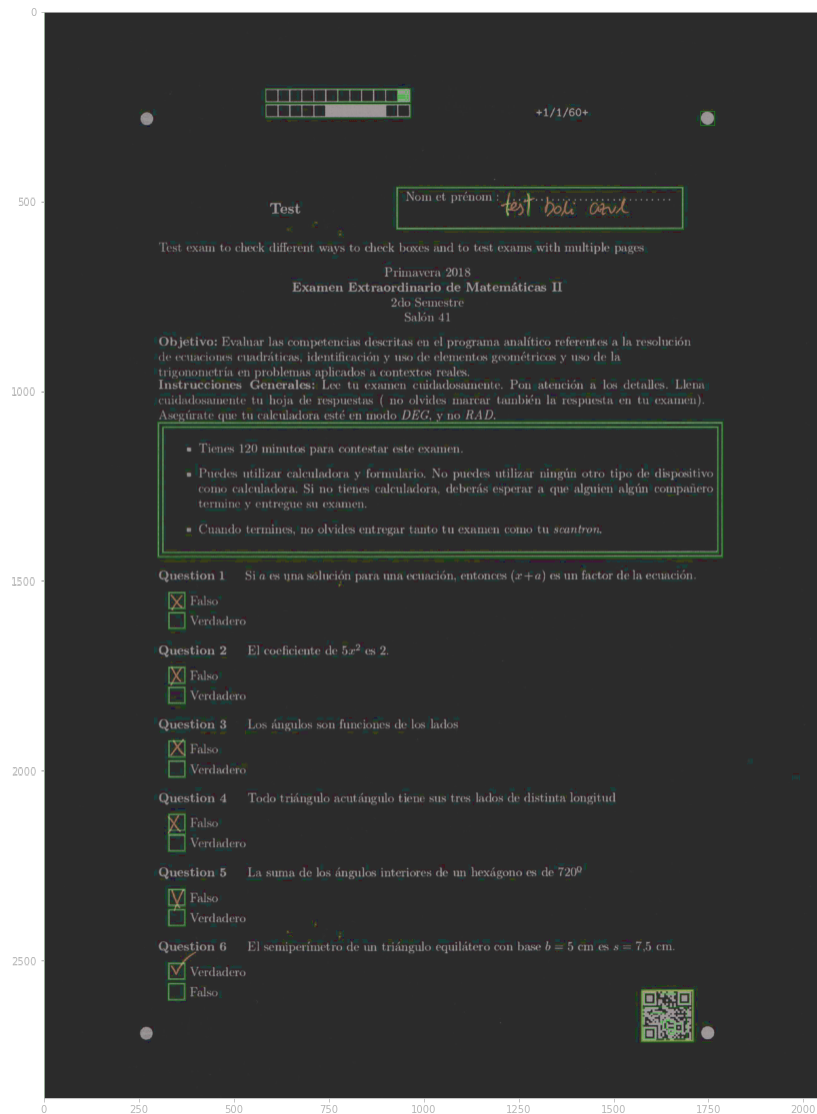


Figure 1.16: Identificación de casillas marcadas



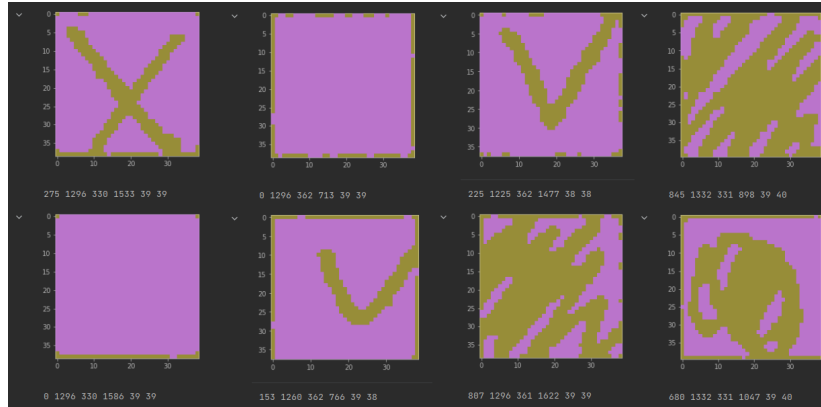


Figure 1.17: Identificación de casillas marcadas

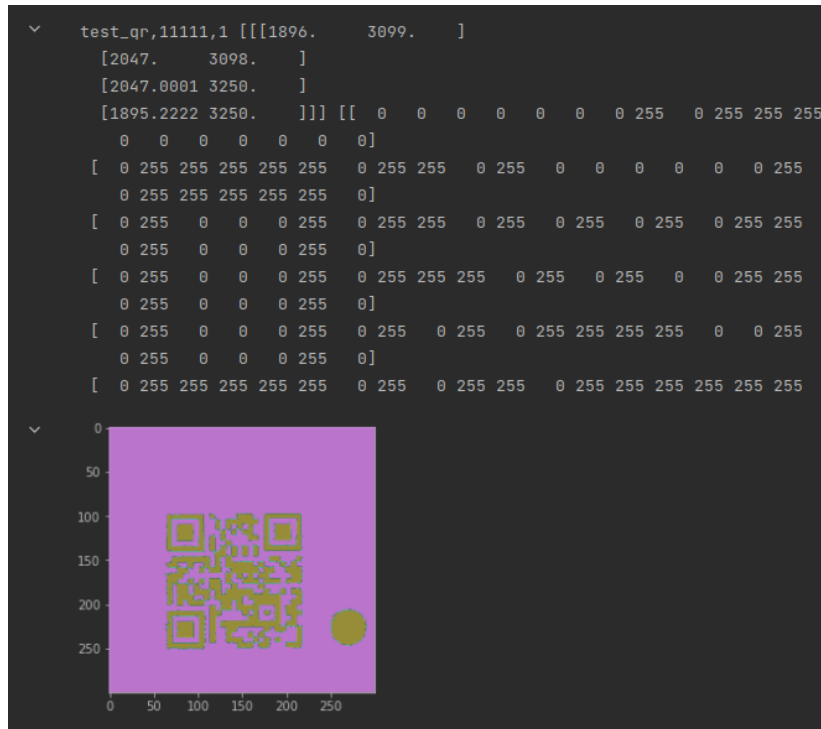


Figure 1.18: Identificación del código QR

## 1 pre-Andrés

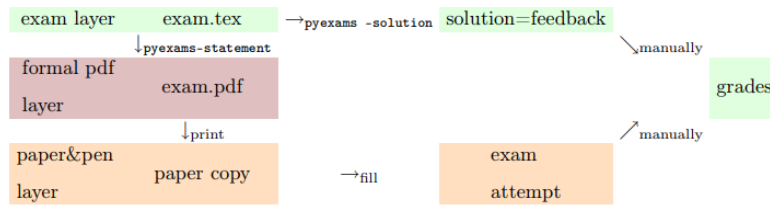


Figure 1.19: Flujo de la aplicación antes

## 2 post-Andrés

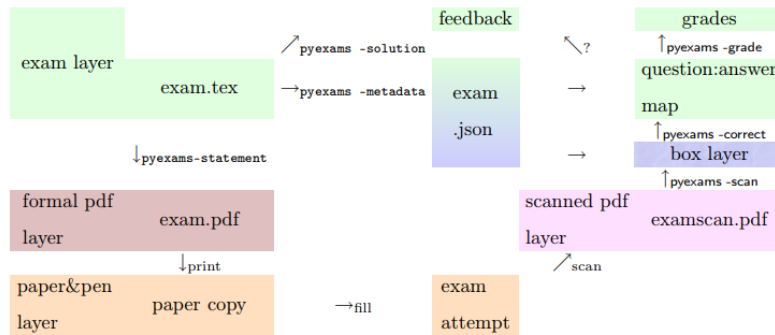


Figure 1.20: Flujo de la aplicación después

- Guardar en booleano si la detección del QR ha funcionado
- Corrección de exámenes; calificación para cada respuesta: vacía (ninguna casilla marcada), nula (numero de casillas marcadas distinto a la solución), correcta e incorrecta
- Diagrama de flujo de los datos en la funcionalidad 'escanear exámenes'.  
Figuras 1.19, 1.20 y 1.21: diagrama por pang del flujo antes y después de la funcionalidad, y de las capas conceptuales

### • 24FEB23 - 01MAR23

- Añadido parámetro '*validation\_th*' a la función *is\_marked()*, parámetro que corresponde al umbral de área rellenada para considerar la casilla marcada en porcentaje, valor por defecto: 0.1
- Guardado el número de página de cada casilla en la función '*analyse*' para exámenes escaneados

### 3 Conceptual layers

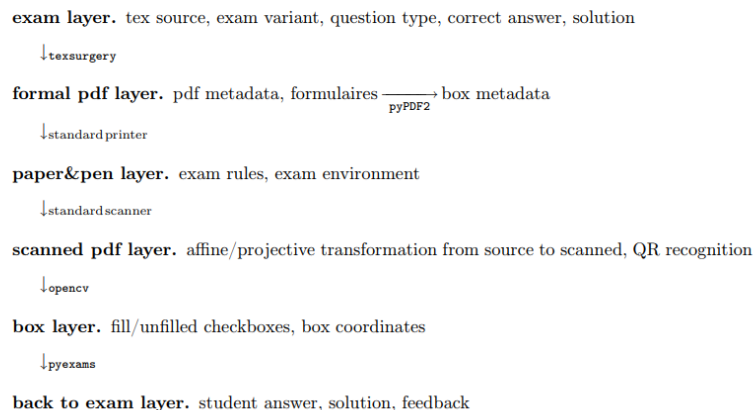
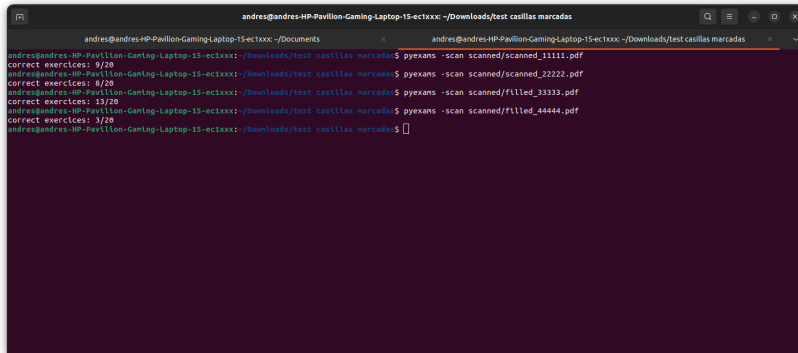


Figure 1.21: Capas conceptuales

- Guardado el número de página de cada casilla en la función '*detect*' para exámenes rellenos. TODO: cambiar también las funciones para enunciados y soluciones generadas
- Posible funcionamiento de corregir orientación: con `cv2.getStructuringElement`, estructura circular y `MORPH.OPEN` -¿ extraer los círculos de AMC, obtener coordenadas y 'corregir' la imagen. Para corregir se puede: a) transformar la imagen modificando los píxeles, b) modificar el sistema de coordenadas para transformar las coordenadas en el examen escaneado a coordenadas rectificadas
- Detección de casillas. Al tener ya las coordenadas de las casillas en el enunciado, identificar las casillas en los exámenes escaneados buscando solo en las zonas cercanas a dichas coordenadas
- ¿Procesar la pagina directamente en vez de guardarla en un png para después leerla?
- 01MAR23, Reu en teams: pang, carlos, angel, sandra
  - El programa creado se puede adaptar para corregir cualquier examen tipo test, no solo generados con pyexams. Definir este caso de uso alternativo. Posibilidad de crear una librería con esa funcionalidad aparte y que pyexams la llame, por temas de cohesión en los comandos de pyexams.
  - Pasos a seguir: terminar el código ( 1 semana); TODO: limpiar bitacora, ampliar estado cuestión y requisitos; empezar trabajo memoria con Sandra ( 4 semanas)
  - demo grabada para la defensa: grabado los pasos sin voz y explicación en vivo



```
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas$ pyexams -scan scanned/scanned_11111.pdf
correct exercises: 9/20
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas$ pyexams -scan scanned/scanned_22222.pdf
correct exercises: 8/20
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas$ pyexams -scan scanned/filled_33333.pdf
correct exercises: 13/20
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas$ pyexams -scan scanned/filled_44444.pdf
correct exercises: 3/20
andres@andres-HP-Pavilion-Gaming-Laptop-15-ec1xxx: ~/Downloads/test casillas marcadas$
```

Figure 1.22: Ejecución con dos exámenes escaneados y dos rellenados

- Función QR no funciona a pang, correos. Fichero sty? Distinta versión python? Función QR sólo funciona con -both -all?
- 01MAR23 - 13MAR23
  - Añadido función `format_scan` que cambia las casillas detectadas al formato json definido
  - lectura de los json de enunciado y de solución
  - añadido función `grade` que compara el json solución con el json enunciado, clasificando cada ejercicio en: empty, null, correct e incorrect
  - probado la ejecución con los dos exámenes escaneados y dos exámenes mas, rellenados con editor pdf (figura 1.22)
- 13MAR23, Reu en navales: pang
  - Identificar qué casilla de OpenCV es qué casilla de pyPDF2 no existe, funciona con fe ciega :) (ambas detectan casillas por el mismo orden de posición, pero en caso de error no funcionaria)
  - Al tener las coordenadas de los 4 puntos del QR, se puede realizar una transformación (TODO: pang)
  - probar detección de qrs con giros de distintos grados
  - si buscas directamente dentro de un rango de coordenadas donde consideras que están las casillas:
    - \* fiabilidad: ancho, coordenadas, giro (tests con examenes girados)
    - \* find contours dentro de un subarray para cada casilla?
  - comprobar si el parámetro de `-scan` es un fichero/carpeta, función para corregir carpetas, throw exception si no existe
- 13MAR23 - X

```
In 470 1 dpi_list = [tr_i[0] for tr_i in transformations]
2 x0_list = [tr_i[1][0] for tr_i in transformations]
3 y0_list = [tr_i[1][1] for tr_i in transformations]
4 alpha_list = [tr_i[2] for tr_i in transformations]
5 scale_list = [tr_i[3] for tr_i in transformations]
6 for i in zip(dpi_list, x0_list, y0_list, alpha_list, scale_list):
7     print(i[0], i[1]/i[0], i[2]/i[0], i[3], i[4]/i[0])

150 2.6010739959611215 11.527334219602729 180.00000000000002 0.01388178807758561
200 2.6048280617172908 11.529164276713429 180.00000000000002 0.013886300195370325
250 2.603311261351402 11.53091296520611 180.00000000000002 0.01389148223874116
300 2.605180408630876 11.53071305086907 180.00000000000002 0.013890078917163299
350 2.6032916693877595 11.52850756867442 180.00000000000002 0.013885143877058782
400 2.604896533823409 11.529371288992671 180.00000000000002 0.013887034520064621
```

Figure 1.23: Relación entre las coordenadas en pyPDF2 y OpenCV, en función del dpi

- 
- Buscado la relación (proporcional al dpi) entre las coordenadas de pyPDF2 y en OpenCV (figura 1.23)

## Chapter 2

# Ideas Sueltas

- Lineas de trabajo
  - Crear una extension en jupyter para pyexams
  - **Ampliar pyexams para escanear exámenes impresos**
    - \* **Obtener las casillas y solución**
    - \* Escanear exámenes y obtener la información de las casillas
    - \* Identificación de casillas marcadas
    - \* Puntuar
- Issues de framagit
  - Issue 1: Choosing all variants from a list instead of random numbers
  - **Issue 2: pyexams `--prepare-send`, pyexams `--prepare-exam`**
  - Issue 3: export to jupyter notebook
  - Issue 4: sort questions in alphabetical order
  - Issue 5: ‘pyexams -send’ receives also extra argument students.csv
  - Issue 6: print the output of pdflatex properly if there is an error
- Documentación del TFG
  - Definir distintos tipos de preguntas que acepta pyexams
  - Diagrama de componentes/clases
  - **Lista de requisitos** -¿ Hacer una entrevista de requisitos en navales
  - Metodologías Ágiles -¿ sacar de la Bitácora y la lista de requisitos los sprints
  - Definir los conocimientos utilizados para la realización del TFG y de donde se ha obtenido
  - **Documentar las pruebas realizadas con imágenes**

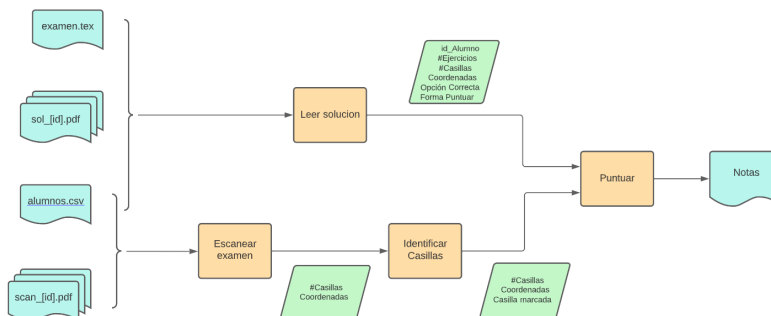


Figure 2.1: Diagrama de componentes inicial

- Utilización de cuadernos jupyter para la realización de pruebas

Componentes de la función Escanear Exámenes:

(Diagrama de componentes inicial, con la herramienta lucid.app en este enlace.)

- Escanear exámenes

*La funcionalidad Escanear exámenes obtiene las siguientes entradas: unos exámenes escaneados en formato [–], el fichero alumnos.csv, los pdf con la solución generada por pyexams y [–]. A partir de estas entradas debe, para cada alumno de alumnos.csv, obtener los datos de la solución, procesar los exámenes escaneados para identificar las casillas, si están marcadas, puntuar los exámenes y generar un archivo con las notas de todos estos.*

- Leer solución: Obtiene de la solución la posición de las casillas y si están marcadas.
- Escanear examen: Obtiene de los exámenes escaneados las casillas
- Identificar casillas: Analiza si las casillas leídas por escanear exámenes están marcadas
- Puntuar: A partir del estado de las casillas de la solución y del examen escaneado puntúa el examen ¿forma de puntuar?

Pruebas de Escanear Exámenes

- Creado un banco de pruebas con el ejemplo de python3/en, añadiendo alumnos
  - Generado los statements y solutions con pyexams
  - Rellenado los statements con un editor de pdf





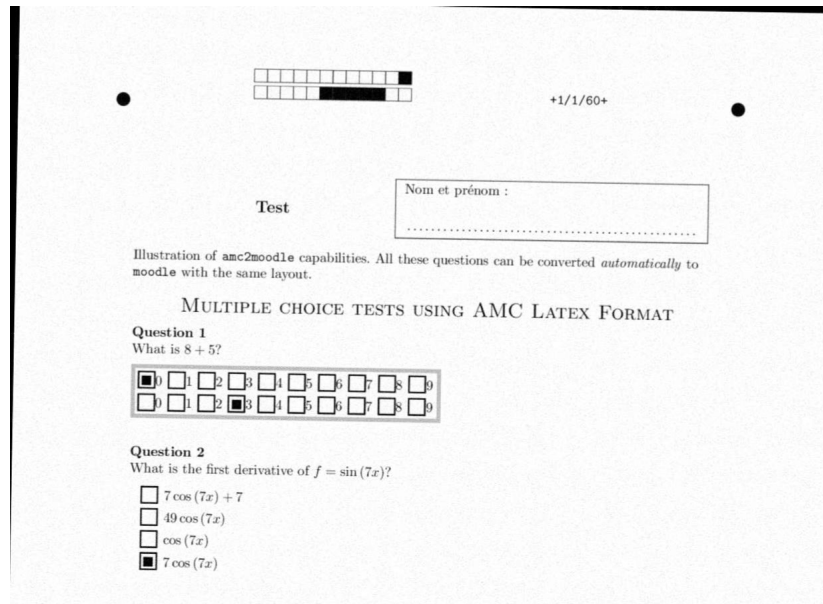


Figure 2.5: Escaneo generado correctamente

- Leer y escribir JSONs desde python
- Exportar paginas pdf a png
-

# Bibliography

- [1] PyExams authors. *Git vcs home page de pyexams*. URL: <https://framagit.org/pang/pyexams>.