

31/10/2025

Herramientas computacionales:
el arte de la programación

TICKET READER

Lector de items de un ticket

PROBLEMA

Trazado granular de costos

En distintos procesos económico-administrativos y en otros ámbitos se requiere de registrar los costos de las compras que se realizan.

Para esto, registrar los costos de los objetos manualmente desde un ticket puede ser tardado, y a nivel contable, caro.

Entonces... >



AUTOMATIZAR

El proceso de extracción de texto es altamente automatizable basándose en modelos de Machine Learning y técnicas de filtrado de imágenes.



Visión computacional

Para la detección de texto en la imagen del ticket



Filtrado de texto

Para identificar las líneas que contienen: ITEM + \${precio}.

```
def preprocess_image(image_path: str) -> Image.Image:  
    """  
        Carga y preprocesa una imagen para mejorar los resultados de OCR.  
  
        El preprocessamiento incluye conversión a escala de grises,  
        umbral adaptativo y eliminación de ruido.  
  
    Args:  
        image_path (str): La ruta al archivo de imagen.  
  
    Returns:  
        Image.Image: Una imagen de objeto PIL preprocesada.  
    """  
  
    # Cargar la imagen usando OpenCV  
    img = cv2.imread(image_path)  
  
    # Convertir a escala de grises, ya que OCR funciona mejor sin color  
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
  
    # Aplicar umbral adaptativo para manejar diferentes condiciones de iluminación.  
    # Esto crea una imagen binaria (blanco y negro) que es más fácil de leer.  
    # 31: El tamaño del bloque de píxeles para calcular el umbral.  
    # 2: Constante C restada de la media (ajuste fino).  
    gray = cv2.adaptiveThreshold(  
        gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  
        cv2.THRESH_BINARY, 31, 2  
    )  
  
    # Aplicar un desenfoque de mediana para eliminar el ruido de "sal y pimienta"  
    # Un kernel de 3x3 es ligero y efectivo para ruido pequeño.  
    gray = cv2.medianBlur(gray, 3)  
  
    # Convertir el array de NumPy (OpenCV) a un objeto PIL (requerido por Tesseract)  
    processed_image = Image.fromarray(gray)  
    return processed_image
```

```
def extract_text_from_image(image_path: str) -> str:  
    """  
        Extrae texto de una imagen usando Tesseract OCR.  
  
    Args:  
        image_path (str): La ruta al archivo de imagen.  
  
    Returns:  
        str: El texto crudo extraido de la imagen.  
    """  
    image = preprocess_image(image_path)  
  
    # Usar Tesseract para convertir la imagen preprocesada en texto  
    # lang="eng" es para inglés. Cambiar a "spa" si el recibo está en español.  
    text = pytesseract.image_to_string(image, lang="eng")  
    return text
```

```
def extract_items_and_resume(text: str) -> tuple:  
    """  
        Analiza el texto crudo del OCR para extraer articulos y un resumen.  
  
        Usa expresiones regulares (regex) para encontrar patrones que coincidan  
        con los articulos (texto + precio) y el resumen (total, impuestos, etc.).  
  
    Args:  
        text (str): El texto crudo de Tesseract.  
  
    Returns:  
        tuple: Una tupla conteniendo (lista_de_items, lista_de_resumen).  
    """  
  
    # Patrón para articulos: (Texto) (digitos) ($precio)  
    # Ej: "PRODUCTO X 1 $10.00"  
    pattern_items = r"([A-Za-z\s]+)\s+\d+\s+(\$\d+\.\d{2})"  
  
    # Patrón para resumen: (Texto) ($precio)  
    # Ej: "SUBTOTAL $20.00"  
    pattern_resume = r"([A-Za-z\s]+)\s+(\$\d+\.\d{2})"  
  
    items = re.findall(pattern_items, text)  
    resume = re.findall(pattern_resume, text)  
  
    return items, resume
```

```
def print_extracted_data(items: list, resume: list):
    """
    Imprime los datos extraídos de forma legible en la consola.

    Args:
        items (list): Lista de tuplas (item, precio).
        resume (list): Lista de tuplas (detalle, total).
    """

    print("■ Artículos encontrados:")
    if not items:
        print("...ninguno.")

    for item, price in items:
        # .strip() limpia espacios en blanco al inicio o final
        print(f"{item.strip()} - {price}")

    print("\n⌚ Resumen:")
    if not resume:
        print("...ninguno.")

    for detail, total in resume:
        print(f"{detail.strip()} - {total}")
```

```
def extracted_data_2CSV(items: list, resume: list):
    """
        Recibe los datos extraidos para guardarlos como 2 archivos CSV:
        uno para Items y otro para resumen.

    Args:
        items (list): Lista de tuplas (item, precio).
        resume (list): Lista de tuplas (detalle, total).
    """

    # 1. Guardar Articulos (Items)
    try:
        # newline='' es necesario para la escritura correcta de CSV
        with open("items.csv", "w", newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(["Item", "Price"]) # Escribir cabecera
            # Escribir datos, limpiando espacios en blanco
            writer.writerows([
                [item.strip(), price] for item, price in items
            ])
        print("\n█ Articulos guardados en 'items.csv'")

    except IOError as e:
        print(f"Error al escribir 'items.csv': {e}")

    # 2. Guardar Resumen
    try:
        with open("resume.csv", "w", newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(["Detail", "Total"]) # Escribir cabecera
            # Escribir datos, limpiando espacios en blanco
            writer.writerows([
                [detail.strip(), total] for detail, total in resume
            ])
        print("█ Resumen guardado en 'resume.csv'")

    except IOError as e:
        print(f"Error al escribir 'resume.csv': {e}")
```

```
def extracted_data_2JSON(items: list, resume: list):
    """
        Guarda los datos extraídos en un único archivo JSON estructurado.

    Args:
        items (list): Lista de tuplas (item, precio).
        resume (list): Lista de tuplas (detalle, total).
    """

    # Convertir listas de tuplas a listas de diccionarios para un JSON más legible
    items_list = [
        {"item": item.strip(), "price": price} for item, price in items
    ]
    resume_list = [
        {"detail": detail.strip(), "total": total} for detail, total in resume
    ]

    # Combinar en un solo diccionario
    data = {
        "items": items_list,
        "resume": resume_list
    }

    # Guardar en archivo
    try:
        with open("receipt_data.json", "w", encoding='utf-8') as f:
            # indent=4 crea un archivo "pretty-printed" (legible)
            json.dump(data, f, indent=4, ensure_ascii=False)
        print("💾 Datos guardados en 'receipt_data.json'")

    except IOError as e:
        print(f"Error al escribir 'receipt_data.json': {e}")
```

```
def main():
    """
    Función principal para ejecutar el flujo completo del script.
    """

    try:
        image_path = "receipt.jpg"
        text = extract_text_from_image(image_path)
        items, resume = extract_items_and_resume(text)

        # 1. Imprimir en consola
        print_extracted_data(items, resume)

        # 2. Guardar en CSV (Llamada a la nueva función)
        extracted_data_2CSV(items, resume)

        # 3. Guardar en JSON (Llamada a la nueva función)
        extracted_data_2JSON(items, resume)

    except FileNotFoundError:
        print(f"Error: No se encontró el archivo '{image_path}'")
    except Exception as e:
        print(f"Ocurrió un error inesperado: {e}")

if __name__ == "__main__":
    main()
```



ÁREAS DE MEJORA

Resolución



No funciona a bajas resoluciones aún sean legibles



Filtros aplicados

Son necesarios varios procesos de imágenes para distintos casos

Limpieza



Las líneas que no detectan correctamente los símbolos son borradas

31/10/2025

Herramientas computacionales:
el arte de la programación

GRACIAS POR SU
ATENCIÓN