Programa Automatizado para Validaciones de Declaraciones Juradas

Programa creado por Cristobal Magnan Grebe
Chile, 2025

Introducción al Programa de Validación de Declaraciones Juradas (DJs)

El programa desarrollado tiene como principal objetivo facilitar y optimizar el proceso de validación de Declaraciones Juradas (DJs) tributarias, documentos críticos emitidos anualmente por el Servicio de Impuestos Internos de Chile (SII.cl). Estas DJs contienen información tributaria detallada que los contribuyentes deben presentar en formatos específicos establecidos por el SII, generalmente proporcionados en archivos Excel directamente desde la página oficial.

Cada Declaración Jurada cuenta con estructuras particulares y requisitos específicos que pueden variar significativamente de un tipo a otro y, además, pueden cambiar anualmente debido a modificaciones en regulaciones tributarias. Ante esta complejidad y variabilidad, este programa se ha diseñado utilizando un enfoque altamente dinámico y modular. Esto significa que el software no solo es capaz de adaptarse automáticamente a diferentes tipos de DJs, sino que también puede acomodarse sin inconvenientes a futuras modificaciones en la estructura de estas declaraciones.

El programa automatiza completamente el análisis de estos documentos mediante la extracción precisa de tablas internas y campos críticos que requieren validación, empleando técnicas avanzadas de procesamiento de datos. El usuario final simplemente debe proporcionar el archivo Excel descargado desde el SII, junto con un archivo TXT previamente preparado con la información específica de la DJ a validar.

El resultado final es una validación visual clara, estructurada y fácilmente interpretable presentada en archivos Excel generados por el programa, destacando de forma explícita las validaciones exitosas y aquellas que requieren atención o corrección adicional.

Gracias a esta herramienta, la tarea de revisar grandes volúmenes de información tributaria se vuelve más eficiente, precisa y menos propensa a errores, permitiendo a los usuarios asegurar una presentación correcta y oportuna de sus obligaciones tributarias.

Programa creado por Cristobal Magnan Grebe
Chile, 2025

Fases del Programa

Fase 1: Extracción y Estructuración Inicial de Datos

La primera fase del programa está dedicada a preparar el terreno para una validación precisa mediante el procesamiento inicial del archivo Excel proporcionado por el SII. Las acciones específicas realizadas en esta fase incluyen:

- **Carga dinámica del archivo Excel**: El software identifica y lee automáticamente todas las tablas internas contenidas en la Declaración Jurada, incluso cuando su número, ubicación o estructura varían considerablemente entre diferentes DJs o versiones anuales.
- **Identificación y clasificación de tablas**: Se determina automáticamente la posición exacta de cada tabla relevante, usando patrones específicos, como el texto "Nº" que habitualmente identifica los encabezados en estos documentos, para asegurar precisión en la extracción.
- **Estructuración automatizada y flexible**: Las tablas extraídas se organizan en formatos de datos estructurados como archivos pickle, facilitando su reutilización eficiente en etapas posteriores.
- **Limpieza preliminar y optimización**: Se ejecuta un proceso de limpieza automatizado que elimina filas vacías, filas con datos irrelevantes o no esenciales (como aquellas que solo contienen números identificadores sin información adicional), asegurando que solo información relevante pase a la siguiente fase.

Esta primera etapa es crucial, ya que establece una base limpia y confiable para realizar las validaciones detalladas en fases posteriores del programa.

Programa creado por Cristobal Magnan Grebe
Chile, 2025

```python
import pandas as pd
import os
import pickle
from openpyxl import Workbook

def contar_tablas_y_organizar(excel_path):
    """Procesa el archivo DJ 1889 Excel y estructura los registros"""
    # Cargar el archivo Excel
    xls = pd.ExcelFile(excel_path)
    sheet_name = xls.sheet_names[0]  # Tomamos la primera hoja
    df = pd.read_excel(xls, sheet_name, dtype=str, header=None)

    # Reemplazar NaN con cadenas vacías
    df.fillna("", inplace=True)

    # Identificar las filas que contienen "Nº" en la columna A
    encabezados = df[df.iloc[:, 0].str.contains("Nº", na=False)].index.tolist()

    # Crear una estructura para almacenar los registros y sus tablas
    registros = {}
    registros_dinamicos = {}

    for i, encabezado in enumerate(encabezados):
        if i < len(encabezados) - 1:
            tabla = df.iloc[encabezado:encabezados[i+1], :].reset_index(drop=True)
        else:
            tabla = df.iloc[encabezado:, :].reset_index(drop=True)

        # Ajustar el encabezado de la tabla
        tabla.columns = tabla.iloc[0]  # La primera fila se convierte en encabezado
        tabla = tabla[1:].reset_index(drop=True)  # Eliminar la fila de encabezado duplicada

        # Modificar el encabezado de las columnas C, D y E con los valores de la primera fila después del encabezado
        nuevos_encabezados = list(tabla.columns)
        nuevos_encabezados[2] = "SUBCAMPOS"  # Asignar nombre fijo a la columna C
        nuevos_encabezados[3] = tabla.iloc[0, 3]  # Asignar el valor de la primera fila como encabezado de la columna D
        nuevos_encabezados[4] = tabla.iloc[0, 4]  # Asignar el valor de la primera fila como encabezado de la columna E
```

```python
def contar_tablas_y_organizar(excel_path):
        nuevos_encabezados[4] = tabla.iloc[0, 4]  # Asignar el valor de la primera fila como encabezado de la columna E
        tabla.columns = nuevos_encabezados
        tabla = tabla[1:].reset_index(drop=True)  # Eliminar la fila duplicada

        # Obtener el número de registro desde la columna Validaciones
        num_registro = tabla.iloc[0, -1].strip()  # Última columna sin espacios
        if num_registro:
            numero_registro = f"Registro{num_registro}"
        else:
            numero_registro = f"Registro{tabla.iloc[1, -1].strip()}.{tabla.iloc[2, -1].strip()}"

        # Limpiar el formato del nombre del registro
        numero_registro = numero_registro.replace("=", "").replace(" ", "")  # Eliminar caracteres no deseados

        # Convertir toda la tabla a string explícitamente antes de analizar filas vacías
        tabla = tabla.astype(str)

        # Eliminar espacios sobrantes en cada celda (inicios y finales)
        tabla = tabla.apply(lambda col: col.str.strip() if col.dtype == "object" else col)

        # — BLOQUE DE ELIMINACIÓN DE FILAS —
        # 1) Filas completamente vacías (todas las celdas vacías)
        filas_vacias = tabla.eq("", axis=1).all(axis=1)

        # 2) Filas donde la columna "CAMPOS" tenga "blanco" o "blancos"
        #if "CAMPOS" in tabla.columns:
         #   filas_blancos = tabla["CAMPOS"].str.strip().str.lower().isin(["blanco", "blancos"])
        #else:
         #   filas_blancos = pd.Series(False, index=tabla.index)

        # 3) Filas que solo tengan contenido en la columna "Nº" y el resto vacío
        if "Nº" in tabla.columns:
            filas_solo_num = (tabla["Nº"].str.strip() != "") & tabla.drop(columns=["Nº"], errors="ignore").eq("", axis=1).all(axis=
        else:
            filas_solo_num = pd.Series(False, index=tabla.index)
        # Combinar condiciones y eliminar filas
```

Programa creado por Cristobal Magnan Grebe
Chile, 2025

```python
def contar_tablas_y_organizar(excel_path):
C:\Users\crist\OneDrive\Escritorio\CARPETAS ORDENADAS\AUTO DJs\__pycache__
        indices_a_eliminar = filas_vacias | filas_solo_num

        tabla = tabla[~indices_a_eliminar].reset_index(drop=True)

        # Almacenar la tabla dentro del registro correspondiente
        if numero_registro not in registros:
            registros[numero_registro] = []
        registros[numero_registro].append(tabla)

        # Guardar en la variable dinámica
        if numero_registro not in registros_dinamicos:
            registros_dinamicos[numero_registro] = []
        registros_dinamicos[numero_registro].append(tabla)

    return registros, registros_dinamicos

def ejecutar_fase1(dj_path, output_dir):
    """
    Ejecuta la Fase 1 del procesamiento.
    - dj_path: Ruta del archivo DJ 1889 Excel
    - output_dir: Carpeta donde se guardarán los archivos generados
    """
    try:
        registros_estructurados, registros_dinamicos = contar_tablas_y_organizar(dj_path)

        # Guardar registros_dinamicos para la fase 2
        registros_path = os.path.join(output_dir, "cmagnanregistros.pkl")
        with open(registros_path, "wb") as f:
            pickle.dump(registros_dinamicos, f)

        # Guardar todos los registros en un único archivo Excel
        output_path = os.path.join(output_dir, "Registros_Estructurados.xlsx")
        wb = Workbook()
        del wb[wb.active.title]  # Eliminar la hoja por defecto
```

```python
def ejecutar_fase1(dj_path, output_dir):
    try:
        registros_estructurados, registros_dinamicos = contar_tablas_y_organizar(dj_path)

        # Guardar registros_dinamicos para la fase 2
        registros_path = os.path.join(output_dir, "cmagnanregistros.pkl")
        with open(registros_path, "wb") as f:
            pickle.dump(registros_dinamicos, f)

        # Guardar todos los registros en un único archivo Excel
        output_path = os.path.join(output_dir, "Registros_Estructurados.xlsx")
        wb = Workbook()
        del wb[wb.active.title]  # Eliminar la hoja por defecto

        for registro, tablas in registros_estructurados.items():
            for idx, tabla in enumerate(tablas):
                sheet_name = f"{registro}_{idx+1}" if len(tablas) > 1 else registro
                sheet_name = sheet_name[:31]  # Limitar a 31 caracteres
                ws = wb.create_sheet(title=sheet_name)

                for r_idx, row in enumerate([tabla.columns.tolist()] + tabla.values.tolist(), start=1):
                    ws.append(row)

        wb.save(output_path)

        print(f"✅ Fase 1 completada. Archivo guardado: {output_path}")
        return output_path  # Devuelve la ruta del archivo para Fase 2

    except Exception as e:
        print(f"❌ Error en Fase 1: {e}")
        return None
```

Programa creado por Cristobal Magnan Grebe
Chile, 2025

El resultado final es una validación visual clara, estructurada y fácilmente interpretable presentada en archivos Excel generados por el programa, destacando de forma explícita las validaciones exitosas y aquellas que requieren atención o corrección adicional.

Gracias a esta herramienta, la tarea de revisar grandes volúmenes de información tributaria se vuelve más eficiente, precisa y menos propensa a errores, permitiendo a los usuarios asegurar una presentación correcta y oportuna de sus obligaciones tributarias.

Fase 2: Validación Detallada y Generación de Informes

En la segunda fase del programa, el enfoque principal es la validación precisa y exhaustiva de la información tributaria contenida en un archivo TXT previamente preparado por el usuario. Esta fase utiliza la información estructurada obtenida en la Fase 1 como base de referencia, garantizando así una correlación precisa entre los datos contenidos en el TXT y las especificaciones exactas proporcionadas en la DJ Excel descargada desde el SII.

Las acciones específicas realizadas en esta fase incluyen:

- **Carga del archivo TXT**: El usuario proporciona un archivo TXT específico, el cual contiene la información real de la DJ a validar. Este archivo debe ser previamente generado por el contribuyente según los lineamientos establecidos por el SII.
- **Extracción y segmentación automatizada de datos**: El programa analiza línea por línea el archivo TXT utilizando algoritmos avanzados, que segmentan automáticamente cada línea según la estructura especificada por la DJ original. Esto implica separar cada dato del archivo según los largos especificados para cada campo en la estructura previamente extraída.
- **Validación rigurosa de formato y contenido**: Cada segmento extraído del archivo TXT se compara detalladamente con las especificaciones originales de validación establecidas en el archivo Excel proporcionado por el SII. Esta comparación se realiza campo por campo para asegurar la integridad y precisión de los datos presentados.
- **Comparación de registros clave (Registro 2 y Registros 3)**: Una validación crítica que realiza el programa es la comparación precisa entre los contenidos del Registro 2 y los Registros 3 del archivo TXT, validando que ambos registros coincidan exactamente, requisito fundamental en muchas declaraciones juradas.
- **Generación automatizada de reportes de validación**: Los resultados de estas validaciones detalladas son almacenados y presentados en archivos Excel de salida. Estos archivos se generan automáticamente y proporcionan al usuario una visualización clara y fácil de interpretar, indicando con exactitud cuáles datos cumplen con las validaciones requeridas y cuáles presentan discrepancias que deben revisarse y corregirse.
- **Adaptabilidad a cambios futuros en validaciones**: Gracias a su estructura dinámica y flexible, esta fase puede adaptarse sin modificaciones significativas a cualquier cambio futuro en las reglas o formatos establecidos anualmente por el SII, extrayendo automáticamente dichas reglas desde el archivo Excel proporcionado.

```python
import pandas as pd
import numpy as np
import os
import pickle
from collections import defaultdict

def ejecutar_fase222(txt_path, registros_path):
    if not os.path.exists(registros_path):
        print("❌ No se encontró el archivo de registros. Asegúrate de ejecutar la fase 1 primero.")
        return []

    # Cargar el diccionario de registros (pickle)
    with open(registros_path, "rb") as f:
        registros_dinamicos = pickle.load(f)

    print(f"✅ Fase 2: Cargados {len(registros_dinamicos)} registros.")

    if not os.path.exists(txt_path):
        print(f"❌ El archivo TXT no existe: {txt_path}")
        return []

    # Leemos todas las líneas del TXT (si bien esto se podría optimizar leyendo en streaming)
    with open(txt_path, 'r', encoding='utf-8') as file:
        lineas = file.readlines()

    resultados = []
    registros_2 = []              # Para construir Registro2_Validacion
    encabezado_registro2 = None    # Encabezado (primeras dos filas) de Registro2
    validaciones_usadas = defaultdict(int)
    validaciones_mapeo = {}

    # Recolectamos validaciones usadas en registros distintos a Registro2
    todas_las_validaciones = set()
    for clave_registro, data in registros_dinamicos.items():
        if "Registro2" not in clave_registro:
            tabla = data[0]
            todas_las_validaciones.update(tabla["VALIDACIONES DE FORMATO"].dropna().unique())
```

Programa creado por Cristobal Magnan Grebe
Chile, 2025

```
def ejecutar_fase222(txt_path, registros_path):
            todas_las_validaciones.update(tabla["VALIDACIONES DE FORMATO"].dropna().unique())

    # Cargamos el archivo de Registros Estructurados (usado para obtener "LARGO DEL REGISTRO")
    registros_estructurados_path = os.path.join(os.path.dirname(registros_path), "Registros_Estructurados.xlsx")
    df_estructurados = pd.read_excel(registros_estructurados_path, sheet_name=None)

    for linea in lineas:
        if len(linea) < 32:
            continue

        # Se extraen el número de registro y de tabla a partir de la línea
        num_registro = linea[:1]
        num_tabla = linea[1:2] if len(linea) > 1 else "1"
        # Se arma la clave: se prueba primero con "RegistroX.Y"; si no existe, se usa "RegistroX"
        posible_clave = f"Registro{num_registro}.{num_tabla}"
        clave_registro = posible_clave if posible_clave in registros_dinamicos else f"Registro{num_registro}"

        if clave_registro in registros_dinamicos:
            tabla = registros_dinamicos[clave_registro][0]

            # ------------------------------------------------------------
            # Caso Registro2: se utiliza la columna "LARGO" de la tabla para determinar
            # los límites de cada campo en la línea TXT, de forma vectorizada.
            # ------------------------------------------------------------
            if "Registro2" in clave_registro:
                # Convertir cada valor en la columna "LARGO" a entero; si no se puede, se asume 0.
                try:
                    largos = tabla["LARGO"].apply(lambda x: int(x) if x and str(x).strip() != "" else 0).tolist()
                except Exception:
                    largos = [0] * len(tabla)
                # Calcular límites acumulados: el primer valor es 0, y cada siguiente es la suma de los anteriores.
                boundaries = np.cumsum([0] + largos)
                # Con estos límites se extrae, para cada campo, el segmento correspondiente de la línea.
                valores_extraidos = [
                    linea[int(boundaries[i]):int(boundaries[i+1])].strip() for i in range(len(largos))
                ]
```

```
def ejecutar_fase222(txt_path, registros_path):
                    linea[int(boundaries[i]):int(boundaries[i+1])].strip() for i in range(len(largos))
                ]
                # Se construye el DataFrame tal como se hacía originalmente:
                df_fila = pd.DataFrame({
                    "CAMPOS": tabla["CAMPOS"].values,
                    "SUBCAMPOS": tabla["SUBCAMPOS"].values,
                    "Información TXT": valores_extraidos
                }).T

                if encabezado_registro2 is None:
                    encabezado_registro2 = df_fila.iloc[:2]

                registros_2.append(df_fila.iloc[2:].reset_index(drop=True))

            # ------------------------------------------------------------
            # Caso para otros registros (Registro1, Registro3, etc.)
            # Se utiliza la lógica original: se recorre cada fila de la tabla y se usa "LARGO" y "CARACTER"
            # para extraer el segmento correspondiente de la línea.
            # ------------------------------------------------------------
            else:
                valores_extraidos = []
                valores_convertidos = []
                posicion = 0

                for _, fila in tabla.iterrows():
                    largo = fila.get("LARGO", "")
                    caracter = fila.get("CARACTER", "")
                    # Se obtiene la validación (aunque en esta versión solo se usa para la visualización)
                    validacion = fila.get("VALIDACIONES DE FORMATO", "")

                    try:
                        largo = int(largo) if largo and str(largo).strip() != "" else 0
                    except ValueError:
                        largo = 0

                    if largo == 0:
```

Programa creado por Cristobal Magnan Grebe
Chile, 2025

```
7   def ejecutar_fase222(txt_path, registros_path):
107                         valores_extraidos.append("")
108                         valores_convertidos.append("")
109                         continue
110
111                     valor_extraido = linea[posicion:posicion+largo]
112                     posicion += largo
113
114                     if caracter == "N":
115                         try:
116                             valor_convertido = int(valor_extraido)
117                         except ValueError:
118                             valor_convertido = ""
119                     else:
120                         valor_convertido = valor_extraido
121
122                     valores_extraidos.append(valor_extraido)
123                     valores_convertidos.append(valor_convertido)
124
125                 df_resultado = pd.DataFrame({
126                     "CAMPOS": tabla["CAMPOS"].values,
127                     "SUBCAMPOS": tabla["SUBCAMPOS"].values,
128                     "Información TXT": valores_extraidos,
129                     "Información TXT (Convertida)": valores_convertidos,
130                     "VALIDACIONES DE FORMATO": tabla["VALIDACIONES DE FORMATO"].values,
131                     "Validar": [
132                         "☐" if pd.notna(valid) and str(valid).strip() != "" else ""
133                         for valid in tabla["VALIDACIONES DE FORMATO"].values
134                     ]
135                 })
136
137                 # Se calcula el largo total exacto de la línea (sin salto de línea)
138                 largo_total = len(linea.rstrip("\n"))
139                 df_largo = pd.DataFrame([
140                     ["Largo"] + [largo_total] + ["" for _ in range(df_resultado.shape[1] - 2)]
141                 ], columns=df_resultado.columns)
```

```
7   def ejecutar_fase222(txt_path, registros_path):
138                 largo_total = len(linea.rstrip("\n"))
139                 df_largo = pd.DataFrame([
140                     ["Largo"] + [largo_total] + ["" for _ in range(df_resultado.shape[1] - 2)]
141                 ], columns=df_resultado.columns)
142
143                 df_resultado = pd.concat([df_resultado, df_largo], ignore_index=True)
144
145                 # Se extrae el valor de "LARGO DEL REGISTRO" desde el archivo de registros estructurados
146                 for sheet_name, df in df_estructurados.items():
147                     if df.shape[1] > 5:
148                         mask = df.iloc[:, 1].astype(str).str.strip().str.upper() == "LARGO DEL REGISTRO"
149                         if mask.any():
150                             valor_largo_del_registro = df.loc[mask, df.columns[5]].values[0]
151                             df_resultado.loc[df_resultado["CAMPOS"] == "LARGO DEL REGISTRO", "SUBCAMPOS"] = valor_largo_del_registr
152
153                 # Se eliminan las filas que contengan la palabra "Blancos" en alguna de las primeras 6 columnas
154                 df_resultado = df_resultado[~df_resultado.iloc[:, :6].apply(
155                     lambda row: row.astype(str).str.contains("Blancos", na=False).any(), axis=1
156                 )]
157
158                 resultados.append((f"{clave_registro}_Validacion", df_resultado))
159
160     # Fuera del bucle, se unen todos los datos de Registro2
161     if registros_2:
162         df_registro2 = pd.concat(registros_2, ignore_index=True)
163         df_registro2 = pd.concat([encabezado_registro2, df_registro2], ignore_index=True)
164         df_registro2.columns = df_registro2.iloc[0]
165         df_registro2 = df_registro2[1:].reset_index(drop=True)
166
167         resultados.append(("Registro2_Validacion", df_registro2))
168
169     return resultados
170
```

Programa creado por Cristobal Magnan Grebe
Chile, 2025

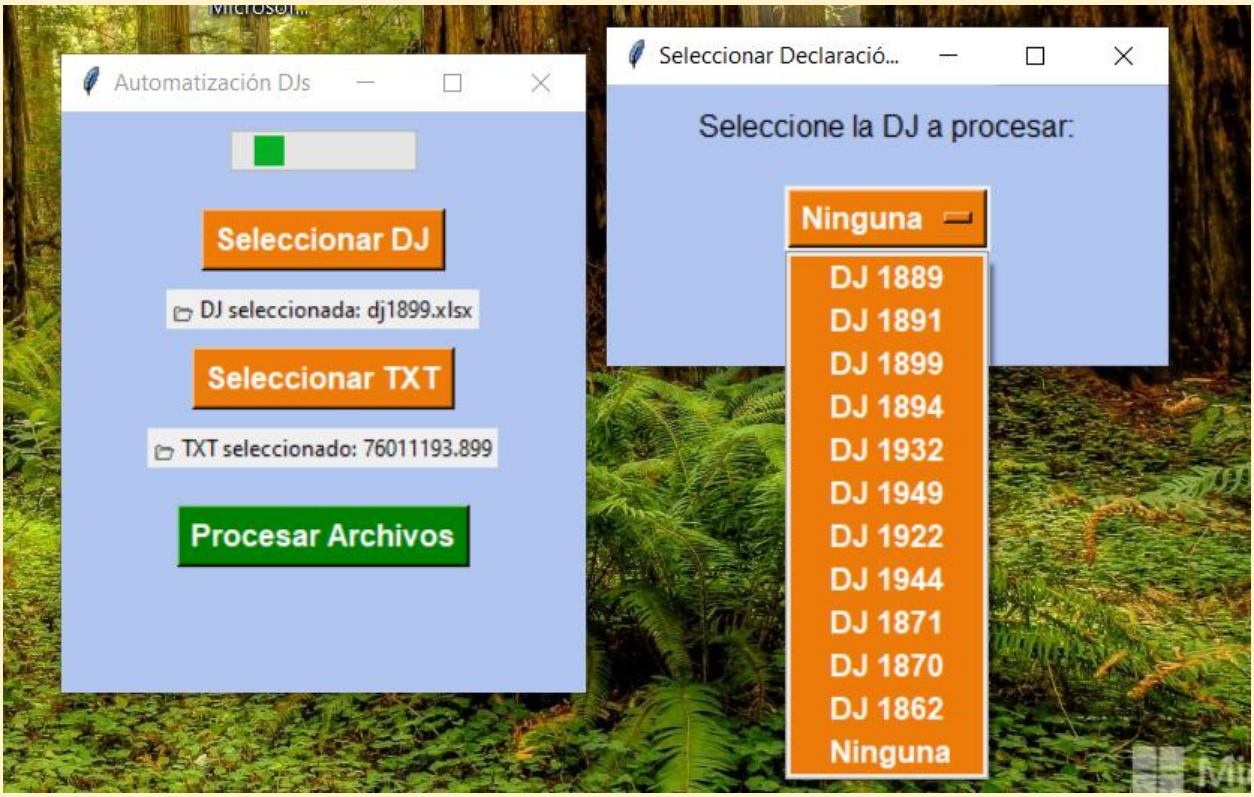
Esta fase proporciona al usuario una herramienta poderosa y precisa que asegura que la información tributaria esté alineada perfectamente con las exigencias normativas, reduciendo significativamente la probabilidad de errores en la presentación final al SII.

Fase 3: Procesamiento Específico y Ajustes Modulares por DJ

La tercera fase del programa aborda directamente la necesidad de adaptarse a la diversidad estructural y de validaciones particulares de cada Declaración Jurada (DJ). Debido a que cada DJ posee reglas específicas y únicas en ciertos aspectos, resulta imposible generalizar completamente todos los procesos de validación en una sola estructura universal. Es por ello que esta fase se ha diseñado con un enfoque modular, permitiendo un manejo individualizado y preciso para cada tipo específico de DJ.

Las acciones específicas realizadas en esta fase incluyen:

- **Interfaz gráfica modular y amigable (Tkinter)**:
  Se ha implementado una interfaz visual clara e intuitiva que permite al usuario elegir fácilmente la Declaración Jurada específica que desea validar mediante un menú desplegable sencillo. Esta interfaz ofrece una experiencia de usuario cómoda y accesible, minimizando errores en la selección.
- **Carga dinámica de scripts específicos por DJ**:
  Cada opción del menú desplegable está asociada a un módulo o script individual, específicamente creado para adaptarse a las particularidades propias de cada DJ. Al seleccionar una opción, el programa ejecuta automáticamente el script correspondiente, asegurando una validación precisa y personalizada.
- **Procesamiento adaptativo y personalizado**:
  Estos scripts específicos realizan ajustes particulares imposibles de generalizar, tales como validaciones adicionales, verificaciones específicas, ajustes en formatos de salida o comprobaciones de campos únicos de la DJ elegida. Esto asegura que cada DJ se trate con la máxima precisión y adaptabilidad a las exigencias tributarias vigentes.
- **Escalabilidad y facilidad de incorporación de nuevas DJs**:
  Gracias al diseño modular y dinámico del programa, incorporar nuevas DJs futuras es un proceso sencillo y ágil. Nuevos módulos específicos pueden agregarse sin afectar la estructura general del programa, permitiendo una rápida adaptación ante cambios o inclusiones de nuevas Declaraciones Juradas por parte del SII.
- **Generación inmediata y personalizada de resultados**:
  Tras la ejecución del módulo específico, el usuario recibe un archivo Excel final claramente estructurado y visualmente detallado, con validaciones específicas aplicadas y resultados claros que resaltan las coincidencias y discrepancias identificadas.



Esta fase garantiza un enfoque especializado y riguroso para cada Declaración Jurada, proporcionando una solución efectiva, precisa y personalizada que asegura un alto estándar de calidad en la presentación y validación de la información tributaria.

Interfaz Gráfica del Programa (GUI)

La interfaz gráfica del programa ha sido cuidadosamente diseñada para ofrecer una experiencia de usuario sencilla, intuitiva y efectiva. Creada utilizando la librería Tkinter de Python, la GUI facilita notablemente la interacción del usuario con las diversas fases del programa, asegurando claridad en cada etapa del proceso de validación.

Las características específicas de la interfaz gráfica incluyen:

- **Selección clara y sencilla de archivos**:
  A través de botones específicos, el usuario puede seleccionar fácilmente tanto el archivo Excel original proporcionado por el SII como el archivo TXT previamente preparado con la información real de la Declaración Jurada. Cada selección es confirmada visualmente, mostrando el nombre del archivo elegido en pantalla para evitar errores de procesamiento.
- **Menú desplegable dinámico para selección de DJ**:
  La GUI incorpora un menú desplegable claramente visible y organizado, desde donde el usuario selecciona la Declaración Jurada específica que desea validar. Esta selección activa automáticamente el script correspondiente diseñado específicamente para las particularidades de esa DJ.

Programa creado por Cristobal Magnan Grebe
Chile, 2025

- **Indicadores visuales y mensajes claros**:
  Durante el proceso de ejecución, la interfaz gráfica proporciona retroalimentación visual clara mediante mensajes emergentes y barras de progreso animadas, indicando claramente en qué fase del proceso se encuentra el programa y alertando al usuario sobre cualquier error o acción adicional requerida.
- **Procesamiento en segundo plano y rendimiento optimizado**:
  Para garantizar fluidez y mantener la interfaz gráfica receptiva en todo momento, el procesamiento principal se ejecuta en segundo plano mediante hilos separados. Esto permite al usuario continuar interactuando con la GUI sin interrupciones ni bloqueos visuales, mejorando considerablemente la experiencia de usuario.
- **Resultado visual estructurado y profesional**:
  Una vez finalizado el procesamiento, la interfaz gráfica genera automáticamente un archivo Excel de salida que es abierto y mostrado claramente al usuario, facilitando la revisión inmediata y la identificación visual rápida de resultados clave y validaciones específicas.

Tecnologías empleadas en la Interfaz Gráfica:

- **Tkinter**: Librería nativa de Python, ampliamente usada para desarrollar interfaces gráficas sencillas y robustas.
- **ttk (Themed Tkinter)**: Utilizada para integrar componentes visuales modernos y amigables como barras de progreso animadas.
- **Multithreading (hilos)**: Implementación avanzada que mejora notablemente el rendimiento y la experiencia del usuario al evitar bloqueos en la interfaz.

Esta interfaz gráfica no solo simplifica el manejo del programa, sino que también mejora significativamente la eficiencia del proceso, permitiendo a los usuarios de cualquier nivel técnico realizar validaciones tributarias complejas de forma cómoda, rápida y efectiva.

Aspectos Importantes a Considerar

- Este programa no genera el archivo TXT; este debe estar previamente preparado según los lineamientos del SII.
- Las validaciones se realizan estrictamente de acuerdo con las especificaciones del archivo Excel descargado del SII, adaptándose automáticamente a cualquier modificación anual en dichas especificaciones.
- La herramienta está diseñada exclusivamente para proporcionar validaciones visuales claras y fácilmente interpretables en un archivo Excel de salida.

Programa creado por Cristobal Magnan Grebe
Chile, 2025

Tecnologías utilizadas:

- **Python**: Lenguaje base para todo el desarrollo y procesamiento lógico del programa.
- **Pandas y Numpy**: Manipulación eficiente y rápida de grandes volúmenes de datos.
- **Openpyxl**: Manejo avanzado de archivos Excel, asegurando precisión y flexibilidad.
- **Tkinter**: Interfaz gráfica amigable, enfocada en una excelente experiencia de usuario.

Este programa representa una solución robusta, adaptativa y eficiente para el manejo de procesos críticos relacionados con las Declaraciones Juradas tributarias, garantizando precisión en la validación, facilitando la tarea de revisión, y asegurando una alta adaptabilidad ante futuros cambios en las estructuras proporcionadas por el SII.

Notas finales sobre la disponibilidad del código fuente

Debido a la extensión considerable del código fuente correspondiente a la interfaz gráfica principal (GUI) y a los múltiples módulos específicos de cada Declaración Jurada en la Fase 3, no se han incorporado en su totalidad dentro de este documento para mantener la claridad y accesibilidad de la presentación.

Sin embargo, todos los archivos de código están disponibles para su revisión individual. Se podrá acceder y descargar cada uno de los módulos específicos según sea necesario para profundizar en su estructura y lógica de funcionamiento.

Además, se incluirán junto a esta presentación los archivos completos de la Fase 1 (fase1.py) y de la Fase 2 (fase222.py), para que estén disponibles de forma inmediata en caso de que se requiera un análisis técnico más detallado o una revisión completa del flujo de procesamiento inicial y de validaciones principales del programa.

Esta disposición busca ofrecer a los interesados un equilibrio entre una presentación visualmente accesible y la posibilidad de acceder al detalle técnico completo de la solución desarrollada.

Programa creado por Cristobal Magnan Grebe
Chile, 2025