

# Universidad de Alcalá Escuela Politécnica Superior

## Grado en Ingeniería Informática

### Trabajo Final de Grado

Encapsulador de contenidos en archivos de texto con  
Python

**Autor:** Raúl Serrano Campillo

**Tutor:** Vera Pospelova

ESCUELA POLITECNICA  
SUPERIOR  
2024



Universidad  
de Alcalá

UNIVERSIDAD DE ALCALÁ  
Escuela Politécnica Superior

**GRADO EN INGENIERÍA INFORMÁTICA**

Trabajo Fin de Grado  
ENCAPSULADOR DE CONTENIDO EN ARCHIVOS DE TEXTO  
CON PYTHON

**Autor:** Raúl Serrano Campillo

**Tutor/es:** Vera Pospelova

**TRIBUNAL:**

**Presidente:** Javier Albert Seguí

**Vocal 1º:** Ana Castillo Martínez

**Vocal 2º:** Vera Pospelova

**CALIFICACIÓN:**

**FECHA:** 08/07/2024

## Contenido

<b>Resumen</b>	<b>5</b>
<b>Abstract</b>	<b>5</b>
<b>Glosario de términos</b>	<b>6</b>
<b>1. Introducción</b>	<b>7</b>
1.1 Introducción al contexto	7
1.2 Descripción del problema	8
1.3 Estado del arte	9
<b>2. Objetivos</b>	<b>11</b>
2.1 Objetivos generales	11
2.2 Objetivos específicos	11
2.3 Estructura del proyecto	11
2.3.1 Estructura de un archivo PDF	11
2.3.2 Desarrollo	12
<b>3. Estructura de un archivo PDF</b>	<b>13</b>
3.1 Estructura general	13
3.1.1 Cabecera	13
3.1.2 Cuerpo	14
3.1.3 Tabla de Referencia Cruzada	14
3.1.4 Tráiler	15
3.1.5 Convenciones léxicas	16
3.2 Estructura del cuerpo	16
3.2.1 Objetos	17
<b>4. Desarrollo</b>	<b>28</b>
4.1 PDF Debugger	28
4.2 Estructura de encapsulación	31
4.2.1 Expresiones regulares	32
4.2.2 Estructuras	33
4.2.2.1 Clases	34
4.3 Carga de los objetos	35
4.4 Diseño funcional	37
4.4.1 Ejecución del programa	37
4.4.2 Menú	38
4.4.3 Opción 1	38
4.4.4 Opción 2	39
4.4.5 Opción 3	41
4.4.6 Opción 4	42
4.4.7 Opción 5	43
4.4.8 Opción 6	44
4.4.9 Opción 7	45
4.4.10 Opción 8	48

<b>5. Conclusiones y trabajo futuro</b>	<b>50</b>
<b>Bibliografía</b>	<b>51</b>
<b>Anexo - Código de la herramienta desarrollado</b>	<b>52</b>

# Resumen

Este proyecto trata de desarrollar una herramienta que encapsule los contenidos de la estructura interna de un fichero PDF en Python. Se abordarán, en un primer lugar, los contenidos teóricos relacionados con conocer la estructura interna de estos archivos y cómo funcionan. En segundo lugar, se explicará en que consiste el proyecto que se ha realizado junto a los resultados obtenidos.

Para ello se ha creado una estructura en Python que almacena algunos de los contenidos que tienen los archivos PDF.

**Palabras clave:** PDF, Python, estructura interna.

# Abstract

This project aims to develop a tool which encapsulates the contents of the internal structure of a PDF file in Python. They will be covered, firstly, the theoretical contents related to understanding the internal structure of these files and how they function. Secondly, it will explain what the project which has been carried out consists of and the obtained result.

**Keywords:** PDF, Python, internal structure.

## Glosario de términos

**PDF:** *Portable Document Format*, formato de archivo para mostrar documentos electrónicamente independientemente del software.

**Python:** lenguaje de programación conocido por su legibilidad y simplicidad.

**Adobe:** empresa de software conocida por sus productos creativos y multimedia.

**RGB:** *Red, Green, Blue*, modelo de color en el que los colores se crean combinando intensidades luminosas de luz roja, verde y azul.

**PostScript:** lenguaje de descripción de páginas desarrollado por Adobe Systems.

**Adobe Illustator:** aplicación de diseño gráfico vectorial desarrollada por Adobe Systems.

**PDF Debugger:** herramienta utilizada para analizar y solucionar problemas en archivos PDF.

**PDL:** *Page Description Language*, lenguaje de descripción de páginas que describe el contenido y el formato de una página de impresión.

**Estructura de árbol:** Estructura de datos jerárquica contenida por nodos conectados.

**Librería:** código compuesto de funciones y datos, presentados en forma de paquete, preparado para ser usado en otros programas.

**Expresiones regulares:** patrones utilizados para encontrar y manipular cadenas de texto basados en reglas sintácticas.

# 1. Introducción

## 1.1 Introducción al contexto

El mundo digital ha transformado la forma en la que interactuamos con la información. Esta información se transmite de múltiples maneras y uno de los formatos más utilizados para la distribución de esa información es el Portable Document Format, más conocido como PDF. En una época de cambios continuos y frecuentes, este tipo de archivo ha evolucionado desde su creación por Adobe Systems[1] en 1993 para convertirse en el formato utilizado actualmente en múltiples ámbitos como la industria, la educación, las finanzas e incluso la vida cotidiana.

El proyecto de la creación de este formato de documento electrónico comenzó con el sueño de una oficina sin papeles. La idea de uno de los fundadores de Adobe, Jhon Warnok, era la de crear un proyecto interno de crear un formato de ficheros para que los documentos se pudieran distribuir por la compañía, viéndose en cualquier ordenador, fuera cual fuera su sistema operativo.

El objetivo de PDF es permitir a los usuarios intercambiar y ver documentos electrónicos de forma fácil y fiable, independientemente del ambiente en el que fueron creados o del ambiente en el que son vistos o impresos.[2]

Para ello, esta empresa contaba ya con 2 tecnologías que encajaban con la idea: El PostScript (Una tecnología independiente del dispositivo que servía para describir documentos) y Adobe Illustrator (Un programa que permitía abrir ficheros PostScript muy simples en plataformas Macintosh y Windows).

De esta forma en 1991 se mencionó por primera vez esta tecnología en una conferencia en California bajo las siglas IPS (Interchange PostScript). Sin embargo, no fue hasta 1993 cuando las tecnologías de crear y mostrar archivos PDF se hicieron públicas dando lugar a la primera versión del fichero PDF. Estos ficheros ya disponían de enlaces internos, marcadores y fuentes, pero únicamente pudiendo utilizar el color RGB.

Con el paso del tiempo se fue mejorando esta primera versión incluyendo soporte para fuentes incorporadas, enlaces internos del documentos y mejoras en la gestión de los colores y la comprensión de imágenes en las 2 siguientes versiones: PDF 1.1 (1994) y PDF 1.2 (1996).

En los siguientes años, se introdujeron las transparencias, las capas y el soporte para el formato gráfico JPEG 2000 junto al soporte para hacer anotaciones y las mejoras en la seguridad con una encriptación de 128 bits en las versiones PDF 1.3 (1999) y PDF 1.4 (2001).

Entre los años 2003 y 2004 (PDF 1.5 y PDF 1.6 respectivamente), se introdujo el soporte de archivos conjuntos, la comprensión de objetos streams, el soporte de incrustación de archivos 3D y mejoras tanto en las anotaciones como en los formularios.

Ya con la versión PDF 1.7 (2006) este formato de documentos fue aceptado como un estándar ISO (ISO 32000-1:2008) y se introdujeron capacidades avanzadas en la encriptación y mejoras en los derechos digitales.

La última versión del PDF es la 2.0, publicada en julio de 2017, en donde mejora a la interoperabilidad entre las distintas implementaciones de PDF, la seguridad, la gestión del color u objetos 3D e incluso las anotaciones y los formularios.

Actualmente, según el tipo MIME (Multipurpose Internet Mail Extensions) detectado en la base de datos de CommonCrawl de julio de 2021, el PDF es el tercer formato más popular en la web (después de HTML y XHTML); más popular que los archivos JPEG, PNG o GIF.[3]

## **1.2 Descripción del problema**

En este mundo cada vez más digitalizado, la capacidad para manejar y procesar información de manera efectiva es crucial. Por ello comprender la estructura de un formato tan utilizado tiene un muy alto valor para quien quiere optimizar una herramienta tan necesaria.

Los archivos PDF contienen una gran variedad de contenido, desde texto simple, hasta imágenes complejas o gráficos incrustados. Sin embargo, la forma en la que se recogen estos documentos en este formato es bastante compleja. Para interactuar con estos documentos de una manera eficiente y efectiva, es fundamental poder encapsular su contenido en una estructura de datos fácilmente manipulable.

En este proyecto se propone el desarrollo de un software que encapsule los contenidos de los ficheros PDF. Para la creación de la estructura se tiene que entender cómo funcionan los archivos PDF, su estructura y cómo están almacenados esos contenidos.

Para ello trataremos de entender, en primer lugar, como están estructurados los archivos PDF y cuáles son las reglas por las que se rigen más allá de la parte visual que es la que el usuario ve.

Seguidamente, trataremos en explicar la forma en la que adaptaremos esa estructura interna de un PDF en una estructura que hayamos creado en Python. En ella podremos generar formas de acceder a todos esos contenidos que tendremos encapsulados y cómo esta estructura trata de asimilarse al funcionamiento de la estructura interna del archivo.

Para el desarrollo del proyecto se hará uso del lenguaje de programación Python[4], para crear la estructura que encapsulará los contenidos del documento. Además, se hará uso de la herramienta PDF Debugger[5] para visualizar esa estructura de un archivo PDF de forma gráfica y poder entender su contenido.

Este trabajo facilita el análisis de la estructura interna del PDF lo que permite comprender la forma en la que se pueden crear y editar archivos que se utilizan en todos los sectores de la actualidad. Además, facilita el uso y la extracción de información de estos archivos, así como, la creación de plantillas de este tipo de archivos.



En resumen, la idea es encapsular los contenidos de un archivo PDF en una estructura de Python. De ella podremos visualizar esos contenidos posteriormente y comprobar el correcto almacenamiento de ellos.

### **1.3 Estado del arte**

En la actualidad y también debido al elevadísimo uso que hay de los documentos con formato PDF, existen varios proyectos o librerías que ayudan a trabajar fácilmente con estos archivos permitiendo crear nuevos documentos o editar otros ya existentes.

#### **Apache PDFBox**

LA biblioteca Apache PDFBox es una herramienta de código abierto en Java para trabajar con documentos PDF. Este proyecto permite la creación de nuevos documentos PDF, la manipulación de documento ya existentes y la habilidad de extraer contenidos de los documentos. Además, contiene varias utilidades de líneas de comando.

Algunas de las características más importantes de esta librería son que permite extraer texto de los archivos, dividir un archivo en múltiples archivos o unir varios archivos PDF en uno solo, extraer datos de formularios PDF o completar un formulario PDF, guardar documentos PDF como archivos de imágenes PNG o JPEG, crear nuevos documentos o firmar digitalmente un PDF.

Además, cabe destacar que esta herramienta la utilizamos en este proyecto para visualizar la estructura interna de un PDF como veremos en la sección 4.1.

#### **PikePDF**

PikePDF es una librería de Python que permite la creación, manipulación y reparo de archivos PDF. Proporciona un envoltorio pasado al lenguaje de Python de la biblioteca de transformación de contenido PDF en C++, QPDF.

Esta biblioteca está orientada a desarrolladores que quieren crear, manipular, analizar, reparar o utilizar el formato PDF. Soporta tanto la lectura como la escritura de estos archivos, incluyendo la creación desde cero. Además, soporta la linealización de archivos PDF, de forma que estos archivos puedan ser descargados y visualizados de forma eficiente en los entornos web, y el acceso a documentos encriptados.

Algunas de las utilidades de esta librería son copiar páginas de un PDF en otros, dividir y unir archivos PDF, extraer contenidos como imágenes de un archivo, reemplazar contenido del archivo sin alterar el resto del archivo o cambiar el tamaño de las páginas o reposicionar contenido.[6]

#### **iText**

iText es una biblioteca de código abierto que sirve para crear y manipular archivos PDF, entre otros, en diferentes lenguajes, aunque es en Java y en .NET los más utilizados. Esta biblioteca permite leer, insertar, actualizar y eliminar cualquier objeto del PDF que se deseen. Así mismo, permite generar un nuevo archivo PDF de forma dinámica a partir de otros archivos PDF, a partir de datos de una base de datos o de datos de un archivo xml o tablas, entre otros.

Es muy utilizado por empresas en la generación e informes a partir de bases de datos, en la facturación electrónica, ya que permite generar facturas u otros documentos financieros en PDF y en la automatización de documentos en las gestiones administrativas entre otros usos.[7]

### **PyPDF2**

PyPDF2 es una librería de código abierto únicamente en Python capaz de dividir, unir, recortar y transformar las páginas de los archivos PDF. También permite añadir datos nuevos, opciones de visualización, contraseñas a los archivos PDF y recuperar texto o metadatos.

Esta herramienta ayuda en la automatización de las tareas administrativas, en el proceso de formularios, en la generación de reportes, así como en la protección de los documentos.[8]

## 2. Objetivos

### 2.1 Objetivos generales

Los archivos PDF siguen una estructura en la que agrupan sus contenidos. Esta estructura es un árbol cuyas ramas tienen los contenidos finales que vemos. Por lo tanto, el objetivo es crear una estructura en Python que encapsule el árbol de contenidos del PDF (/StructTreeRoot) y las referencias objeto de ese árbol. De esta forma podremos visualizar los contenidos e incluso poder modificarlos sin corromper el archivo.

En ese árbol habrá que distinguir entre los distintos elementos definidos en objetos. Estos objetos podrán contener rutas hacia próximos objetos formando esta estructura de árbol o podrán ser las puntas finales del árbol, las hojas, en las que se encuentran los elementos que finalmente visualizamos.

La idea es encapsular todos esos contenidos en la estructura que creamos y que sea accesible, es decir, que podamos acceder a esos contenidos junto a sus características, pudiendo luego visualizarlos e incluso modificarlos.

### 2.2 Objetivos específicos

1. Conseguir estructurar los elementos del PDF, los cuales se encuentran encapsulados en objetos, consiguiendo almacenar el árbol que estructura el PDF.
2. Ubicar los elementos de mayor importancia que vamos a utilizar en el trabajo
3. Almacenar las etiquetas finales de la estructura en la que se almacenan las instrucciones finales del archivo. En la mayoría de los casos, estas instrucciones serán la de imprimir un texto.
4. Almacenar los diferentes atributos que contenga el PDF para poder acceder a ellos en caso de querer modificarlos.
5. Generar un nuevo archivo txt con los mismos contenidos del PDF, pero con las nuevas modificaciones que se hayan realizado en la estructura del archivo.

### 2.3 Estructura del proyecto

En este proyecto se abarca el análisis de la estructura de un archivo PDF y el propio desarrollo de la herramienta realizada. A continuación, se describen las distintas secciones que nos podemos encontrar en el trabajo.

#### 2.3.1 Estructura de un archivo PDF

En esta sección se definirán las bases teóricas del proyecto. Se describirá la estructura interna que tienen los archivos PDF explicando desde la estructura más general del documento a sus contenidos básicos y como se relacionan.

### 2.3.2 Desarrollo

En esta sección se describirá el desarrollo de la herramienta que se ha realizado en este proyecto. Para ello se explicará una herramienta que ayuda a entender la estructura interna de los ficheros PDF y se detallará en qué se basa la estructura que se ha creado para almacenar estos documentos y cómo se genera. Finalmente, se describirá cómo funciona la herramienta y qué utilidades tiene esta.

### 3. Estructura de un archivo PDF

Como hemos definido anteriormente, los archivos PDF constan de una estructura interna en la que se definen los formatos, posiciones, contenidos, instrucciones... de los cuales el archivo está compuesto.[9]

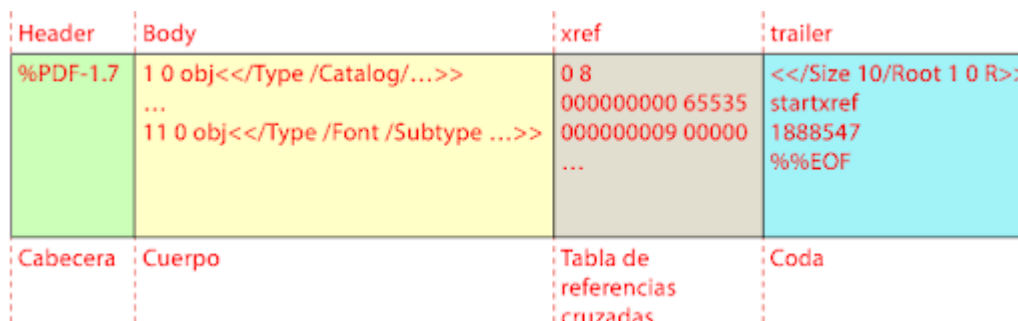
Sin embargo, esta estructura interna no se ve al abrir un archivo PDF. En ese caso solo veríamos el PDF en su forma preparada para la vista de un lector. Para poder acceder a la estructura interna del PDF es necesario cambiar la extensión del archivo a un archivo txt.

Además, normalmente los archivos vienen codificados y es necesario hacer uso de herramientas externas que nos permiten decodificar el archivo para poder ver esa estructura interna a la que nos referimos.

En las secciones que se presentan a continuación se detallará la estructura interna que tienen estos archivos y se describirán los diferentes elementos de los cuales se componen.

#### 3.1 Estructura general

Al hablar de la estructura de un PDF tenemos que entender que esta es un archivo de código está compuesta por 4 partes obligatorias en todos los archivos con este formato: la cabecera, el cuerpo, la tabla de referencia-cruzada y el tráiler.



*Ilustración 1: Esquema de la estructura interna de un PDF.*

Fuente: [http://www.gusgsm.com/la\\_estructura\\_de\\_los\\_documentos\\_pdf](http://www.gusgsm.com/la_estructura_de_los_documentos_pdf) [10]

##### 3.1.1 Cabecera

La cabecera (Header) de un archivo PDF es únicamente la primera línea del archivo. Esta sirve para identificar el archivo con la extensión de un PDF y para especificar la versión del formato PDF que está utilizando dicho archivo. La cabecera sigue siempre la misma estructura en la que podemos obtener el identificador del archivo como el formato propuesto de la forma '%PDF' seguido de la versión del PDF compuesta por dos números separados por un punto como puede ser '1.6' (versión 1.6 de PDF). Así una posible cabecera para un archivo PDF sería '%PDF-1.6' estando esta, en la primera línea del archivo. Esta línea es completamente necesaria para que cualquier software pueda interpretar el archivo correctamente.

## %PDF-1.6

*Ilustración 2: Cabecera de un PDF.*

*Fuente: propia*

### 3.1.2 Cuerpo

El cuerpo (Body) es la parte más extensa y compleja del archivo. Esta parte está dividida en estructuras de datos llamadas objetos en las que se describe el contenido que se va a mostrar. Así pues, estos objetos contienen referencias a las páginas del archivo junto a sus dimensiones y los distintos recursos como fuentes e imágenes que se utilizan en el archivo. Además, también contienen los flujos de contenido con las estructuras e instrucciones de imprimir los textos que posteriormente se muestran. En esta parte del archivo es donde se ha profundizado el trabajo y se explicará con más detalle en la sección 3.2.

### 3.1.3 Tabla de Referencia Cruzada

La tabla de referencia-cruzada (Cross-Reference Table) es una parte crucial que actúa como índice del archivo que permite leer a los visualizadores de PDF localizar rápidamente los objetos dentro del archivo sin tener que leer el archivo al completo. Esta tabla permite acceder rápidamente a cualquier objeto del archivo, facilita la edición de los archivos sin tener que reescribir todo el archivo y asegura que el PDF está bien estructurado.

En primer lugar, la tabla de referencia-cruzada contiene una cabecera que es 'xref' lo que indica en el código que nos encontramos en esta parte del archivo.

En segundo lugar, nos encontraremos una nueva línea en la que se muestran las secciones de la tabla compuesta por dos números. El primer número indica el objeto en la sección (este suele ser 0) y el segundo el número de secciones que contiene la tabla. Por ejemplo, '0 2934' indica que hay 2934 secciones de la tabla que además se corresponde con el número de objetos que hay en el archivo.

Por último, la tabla contiene las propias entradas de objetos. Indicando, primeramente, el desplazamiento en bytes y después el número de generación del objeto, que indica la versión del objeto.

```
xref
0 2934
0000000000 65535 f
0000000015 00000 n
0000000531 00000 n
0000004171 00000 n
0000004192 00000 n
```

*Ilustración 3: Tabla de referencia-cruzada de un PDF.*

*Fuente: propia*

Así pues, en el ejemplo mostrado, el objeto 0 está libre y tiene un número de generación 65535 lo que indica que es una entrada especial y no se usa. Sin embargo, el objeto 1 comienza en el byte 15 del archivo, tiene un número de generación 0 (versión 0 del objeto) y está en uso (n). Cabe destacar que esta tabla, que se muestra en la ilustración,

está cortada puesto que tiene 2934 entradas como indica en la cabecera de la tabla de referencia-cruzada.

### 3.1.4 Tráiler

El tráiler (Trailer) de un archivo PDF es la sección que sigue a la tabla de referencia-cruzada. Esta parte contiene información necesaria sobre la estructura y elementos del documento. Este permite encontrar rápidamente la tabla de referencia cruzada y ciertos objetos especiales.

El tráiler está compuesto por varias partes. En primer lugar, contiene una cabecera que indica el comienzo de la sección con la palabra clave ‘trailer’.

En segundo lugar, contiene un diccionario. Esto son estructuras de almacenamiento de datos que almacenan pares de claves y valores. Se detallará más en la explicación de estas estructuras que nos encontramos en los objetos que se detallan en la sección 3.2. Dependiendo del archivo, este diccionario podrá contener una información u otra. Sin embargo, las claves ‘Size’ y ‘Root’ son obligatorias en esta sección indicando el número total de entradas de la tabla de referencia-cruzada y la referencia indirecta al objeto de catálogo del documento respectivamente. Además, puede contener otras claves opcionales como ‘Info’, que muestra la referencia al objeto con la información del documento; ‘ID’, que muestra un par de identificadores únicos para el documento; o ‘Encrypt’ que muestra si el documento está cifrado.

Finalmente, contiene la línea ‘startxref’ y en la siguiente línea un número, el cual indica el byte del archivo donde comienza la tabla de referencia-cruzada por lo que es una referencia indirecta a esta tabla. Además, la sección termina con la línea ‘%%EOF’ que marca el final del archivo PDF.

```
trailer
<<
/Length 50
/Info 6 0 R
/Root 1 0 R
/ID [<647B02643A16A54C89E7FDE6C69746DC> <81FC2D65EAE979418F99CAE9DBD8B0B9>]
/Size 2934
/Type /XRef
/W [1 3 1]
/Index [2 1 169 1 172 1 175 1 3052 1
3069 3]
/DecodeParms <<
/Columns 5
/Predictor 12
>>
/Filter /FlateDecode
>>
startxref
1135388
%%EOF
```

*Ilustración 4: Tráiler de un PDF.  
Fuente: propia*

### 3.1.5 Convenciones léxicas

Para entender el código que estructura estos archivos es necesario entender que los archivos PDF son una secuencia de bytes de 8 bits. Los caracteres usados en este código pueden ser agrupados en tokens como palabras claves o números que son posteriormente interpretados.[11]

Algunas reglas generales se aplican sobre el cuerpo del archivo, y frecuentemente en otros lenguajes, en un archivo PDF. Así pues, existen 3 tipos de caracteres: los caracteres regulares, los espacios en blanco y los delimitadores.

- Los espacios en blanco pueden ser de varios tipos dependiendo del código de carácter que tenga. Entre ellos podemos encontrar el espacio en blanco, el salto de línea, el salto de página... Los tipos de salto están definidos en la siguiente tabla representando también su código de carácter.

Character code	Meaning
0	Null
9	Tab
10	Line feed
12	Form feed
13	Carriage return
32	Space

*Ilustración 5: Tabla de espacios en blanco.*

*Fuente: J. Whittington, PDF explained, 1st ed. Sebastopol, Calif: O'Reilly Media, 2011*

- Los delimitadores son ( ) < > [ ] / %, y normalmente se usan para definir arrays, diccionarios o distintas estructuras que podemos encontrar en el código del archivo.
- El resto de los caracteres son los caracteres regulares sin ningún significado especial.

Además, los archivos PDF puede usar <CR>, <LF>, o una secuencia <CR><LF> para terminar una línea.

## 3.2 Estructura del cuerpo

El cuerpo del archivo contiene la estructura más compleja y extensa del archivo. Es importante entender que esta parte está compuesta por los distintos objetos que conforman el contenido. Estos objetos pueden ser de distintos tipos como páginas, fuentes, imágenes, anotaciones, etc.

Estos objetos están relacionados unos con otros formando una estructura de árbol en la que unos objetos son hijos de otros. Esta estructura es la que hace posible que se haga referencia a las páginas y sus contenidos estando estos almacenados de una forma muy específica.



### 3.2.1 Objetos

Los objetos son las estructuras del cuerpo de un archivo PDF que describen el contenido y la estructura del documento. Dentro de un archivo PDF son las estructuras fundamentales para almacenar la información.

La estructura clara y definida de los objetos facilita mucho la creación, manipulación e incluso la visualización de un archivo de una forma más precisa y eficiente.

El archivo al completo está compuesto de estas estructuras que pueden variar según el tipo, pero todas empiezan por una cabecera en la que el primer carácter que se muestra es el identificador del objeto. Este identificador será el que puedan utilizar otros objetos para llamar a este creando la estructura de objetos enlazados.

#### 3.2.1.1 Tipos de objetos

Aunque siguen una estructura común, hay diferentes tipos de objetos cada uno con sus propias funciones y usos en el documento[12].

```
4 0 obj
<<
/Count 20
/Kids [7 0 R 8 0 R 9 0 R 10 0 R]
/Type /Pages
>>
endobj
```

*Ilustración 6: Objeto de un PDF de tipo páginas.*

*Fuente: propia*

Además de un identificador, que se encuentra al comienzo del objeto, los objetos tienen una referencia del 'Type'. Esta referencia el tipo de objeto que puede ser este. Alguno de estos tipos pueden ser:

#### **Catálogo (/Catalog)**

Los objetos de tipo catálogo actúan como el punto de entrada principal del documento. Además, estos objetos contienen referencias a otros objetos esenciales en el archivo como pueden ser los objetos que hacen referencia a páginas.

Estos objetos tienen como función servir como nodo raíz del árbol de objetos. Son la referencia principal a las páginas del documento, así como a otros recursos globales y permiten la navegación del PDF.

#### **Páginas (/Pages)**

Los objetos de tipo páginas crean una estructura jerárquica de las páginas del documento. Crean una estructura de árbol entre otros objetos de tipo páginas o directamente los objetos de tipo página.

Estos objetos tienen como función agrupar las páginas de una forma jerárquica de forma que se facilite la navegación y manipulación del documento. Además, puede contener propiedades que se apliquen a todas las páginas hijas de estos objetos.

**Página (/Page)**

Los objetos de tipo página representan cada uno a una única página del documento. Por lo tanto, existirán tantos objetos de este tipo como páginas tenga el archivo PDF. Cada objeto contiene información sobre el contenido de la página como pueden ser su tamaño, los recursos utilizados o los contenidos que esta tiene.

Es importante distinguir que los objetos de tipo páginas y de tipo página no son el mismo. El primero se refiere a varias páginas, no a uno concreto. Sin embargo, los objetos de tipo página se refieren únicamente a una sola página. Además, estos últimos contienen la referencia al objeto de tipo contenido de una página.

**Contenidos (/Contents)**

Los objetos de tipo contenido contienen el contenido real que se muestra en una página específica. Los contenidos definen cómo se presentan los elementos visuales como texto, imágenes o gráficos de una página. Al ser de una sola página estos objetos están referenciados en los objetos de tipo página.

Estos objetos proporcionan un flujo de datos que contiene unas instrucciones en el lenguaje de descripción de página (PDL). Un visor de página es el que interpreta estas instrucciones para renderizar las páginas. Estos visores de páginas permiten mostrar, navegar y visualizar el contenido de las páginas del PDF convirtiendo el contenido del PDF en texto o imágenes. Además, permiten desplazarse por las páginas del documento o hacer zoom en ellas.

**Recursos (/Resources)**

Los objetos de tipo recursos contienen los recursos necesarios para renderizar el contenido de las páginas del documento. Estos recursos pueden ser fuentes de texto, imágenes, gráficos colores u otros elementos.

La función de estos objetos es organizar y proporcionar un acceso eficiente a todos los recursos que se necesitan para poder visualizar correctamente una página de un archivo PDF. Además, estos recursos pueden compartirse entre múltiples páginas para optimizar el tamaño del archivo y mejorar el rendimiento del procesamiento del PDF.

**Fuente (/Font)**

Los objetos de tipo fuente representan las fuentes tipográficas utilizadas en el texto de las páginas. Estos objetos definen esas fuentes y en las instrucciones de texto se referencian estas fuentes. Cada objeto de este tipo define una fuente que puede ser utilizada para mostrar caracteres y símbolos en diferentes estilos tamaños o pesos dentro del documento.

**Imágenes (/XObject)**

Los objetos de tipo imagen representan aquellas imágenes que se utiliza en las páginas del documento. Estos objetos permiten la inclusión de gráficos y fotografías dentro del PDF, siendo una solución a la hora de representar contenido visual complejo. Tanto los objetos de tipo fuente como estos se representan en el diccionario que contienen los objetos de tipo recursos.

La forma en la que estos objetos representan las imágenes es definiendo algunas características de la imagen como la altura o el ancho y posteriormente almacenando los datos de la imagen en un stream codificado.

Estos objetos son, posteriormente, renderizados usando posiciones de trazas (path positions) en las que se configura una matriz de transformación y se define la trayectoria del gráfico antes de incorporarlo a la página. Sin embargo, este proceso se encuentra fuera del alcance de este trabajo.

### 3.2.1.2 Estructura de un objeto

Cada objeto tiene una estructura definida que incluye varias partes: un identificador único, un tipo, un conjunto de atributos (claves) y un contenido asociado.

El identificador único lo podemos encontrar al comienzo de un objeto. La primera línea indica el comienzo del objeto teniendo, en primer lugar, el identificador único seguido de la versión de generación y después la palabra clave ‘obj’. Así en la ilustración 7 el identificador único sería 1 y la versión de generación del objeto sería 0.

```
1 0 obj
<<
  /Lang (en-US)
  /MarkInfo <<
    /Marked true
    /Suspects false
  >>
  /Metadata 2 0 R
  /Outlines 3 0 R
  /Pages 4 0 R
  /StructTreeRoot 5 0 R
  /Type /Catalog
  /ViewerPreferences <<
    /DisplayDocTitle true
  >>
  >>
endobj
```

*Ilustración 7: Objeto de un PDF.*

*Fuente: propia*

En segundo lugar, tenemos un diccionario de referencias. Este diccionario está delimitado por los caracteres ‘<<’ y ‘>>’. En este diccionario encontramos unas duplas clave-valor en las que la clave comienza por el carácter ‘/’ y va seguido de un espacio en blanco que la separa del valor.

Por ejemplo, si nos fijamos en el objeto 1 podemos ver que la clave ‘Lang’, que hace referencia al idioma del PDF (Language), tiene como valor (en-US). Es decir, que el idioma que tiene este archivo es inglés, concretamente de Estados Unidos (US).

Finalmente podemos ver en la ilustración que la palabra clave ‘endobj’ es la que indica el final del objeto.

### 3.2.1.3 Tipos permitidos en objetos

Dentro de los objetos podemos encontrar distintos tipos de valores que soportan estas estructuras. Podemos distinguir estos valores en tipos básicos, tipos complejos y referencias. Los tipos que soportan pueden ser los siguientes.

- Números enteros y números reales como pueden ser 42 o 3.1415 respectivamente.
- Strings o cadenas de texto, las cuales se encuentran encapsuladas por paréntesis ( ) y que pueden venir en varias codificaciones. Un ejemplo sería (The Quick Borwn Fox).
- Nombres, que se utilizan como claves de los diccionarios entre otras cosas. Estos siempre comienzan por el carácter '/', por ejemplo, /Blue
- Valores booleanos, los cuales están delimitados por las palabras clave true y false.
- El tipo null o tipo vacío, que está denotado por la palabra clave null.

Otros tipos compuestos son:

- Arrays o listas que contienen una colección de otros objetos y que están delimitados por corchetes [ ], por ejemplo, [1 0 0 0].
- Diccionarios, que consisten en una colección desordenada de parejas nombres y un tipo de objeto definiéndose como clave-valor. Están delimitados por los caracteres << y >>. Por ejemplo <</Contents 4 0 R /Resources 5 0 R>> en él se enlaza la clave 'Contents' con la referencia '4 0 R' y la clave 'Resources' con la referencia '5 0 R'.
- Streams, que contienen datos binarios, junto a diccionarios, describiendo atributos de los datos tal como la longitud o parámetros de compresión. Estos se utilizan para almacenar imágenes, tipos de fuente... Además, el comienzo y el fin de estos objetos están representados por las palabras clave 'stream' y 'endstream' respectivamente; y contenidos entre los caracteres << y >>.

### 3.2.1.4 Estructura de árbol

Por lo tanto, podemos ver que los objetos almacenan información que puede ser utilizada posteriormente. Sin embargo, las formas entre las que se relacionan los objetos son un poco más variadas.

Recordemos que, como hemos mencionado anteriormente, los objetos crean una estructura de árbol en la que se relacionan unos objetos con otros agrupando los contenidos de una forma más eficiente para acceder a ellos.

La forma de hacer una referencia a un objeto es mediante la referencia en la que se indica el número de identificación del objeto seguido de la versión de generación del objeto y finalmente del carácter 'R' que indica que es una referencia.

Así, por ejemplo, en la ilustración 7, se puede ver que la clave 'Pages' hace referencia al objeto con número de identificación 4. Además, este objeto es de tipo páginas por lo que sigue el árbol de las páginas.

```

41 0 obj
<<
/K [1242 0 R 534 0 R 535 0 R 536 0 R 537 0 R 538 0 R 539 0 R 540 0 R 541 0 R 542 0 R
543 0 R 544 0 R 545 0 R 546 0 R 547 0 R 548 0 R 549 0 R 550 0 R 551 0 R 552 0 R
553 0 R 554 0 R 555 0 R 556 0 R 557 0 R 558 0 R 559 0 R 1243 0 R 566 0 R 567 0 R
568 0 R]
/P 11 0 R
/S /NonStruct
/T (Page 9)
>>
endobj

```

*Ilustración 8: Objeto de un PDF.  
Fuente: propia*

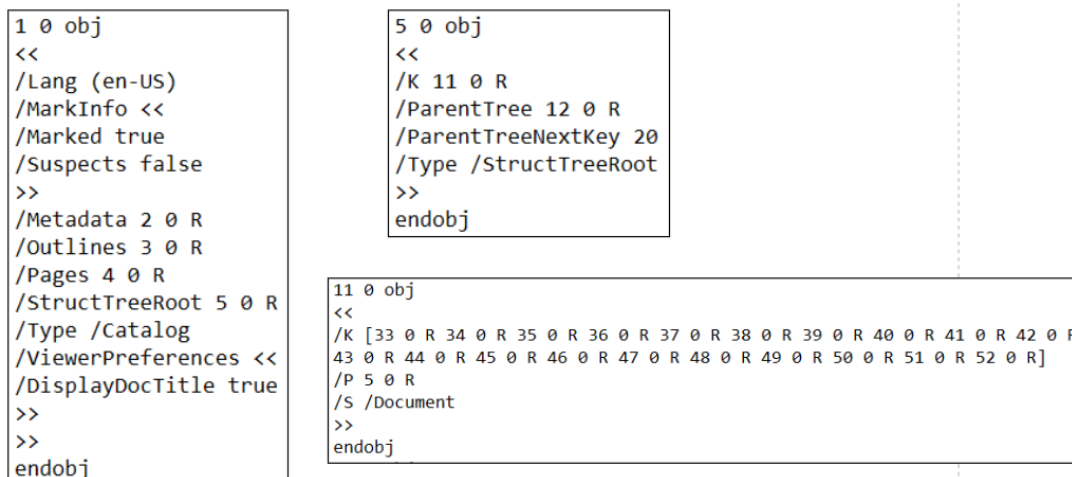
Por lo tanto, podemos referenciar a objetos que siguen la estructura de árbol desde las claves indicando la información que van a contener los objetos referenciados. Sin embargo, la forma más utilizada para referenciar a otros objetos en la estructura de árbol es mediante la declaración de las claves Kids y Parent.

Por ejemplo, fijándonos en la ilustración 8, la clave 'P', que se refiere al objeto padre (Parent) de este objeto, tendría un valor '11 0 R' que hace referencia al objeto 11. Es decir, que el padre del objeto 41 sería el objeto 11.

Por otro lado, la clave 'K' hace referencia a los hijos (Kids) de este objeto. Al tener varios hijos en vez de tener una referencia, lo que hay es un array de referencias. Con lo cual, los hijos del objeto 41 serían los objetos con identificadores 1242, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567 y 568.

De esta forma, se crea una estructura de árbol. Sin embargo, es importante destacar que dentro de la estructura existe un árbol principal referenciado en primer lugar por un objeto de tipo /StructTreeRoot. Los hijos de este objeto son los que formarán esa estructura de árbol que finaliza en los contenidos de las páginas.

Por lo tanto, en el primer objeto del PDF se referencia al objeto de tipo /StructTreeRoot. Este a su vez referencia en sus hijos al objeto que seguirá formando esa estructura de árbol principal que a su vez tendrá nuevos hijos que seguirán formando esa estructura.



*Ilustración 9: Objetos que definen el /StructTree.  
Fuente: propia*

En este ejemplo podemos ver que el objeto con identificador 1 define que el objeto /StructTreeRoot es el objeto con identificador 5. Este último, a su vez, define su hijo que a su vez define otros hijos definiendo el árbol de estructura.

Finalmente, podemos referirnos al /StructTreeRoot como si fuera el tronco principal de todo el árbol que se va ramificando hasta llegar a los objetos finales u hojas que serían los objetos que contienen el propio contenido de las páginas.

### 3.2.1.5 Objetos de contenido

Hasta el momento hemos visto que los objetos están almacenando información sobre los contenidos y que siguen una estructura. Sin embargo, cuando pensamos en un archivo PDF pensamos en un documento en el que hay texto impreso. Esos textos están almacenados en estos objetos de contenido mediante instrucciones.

Cuando analizamos un objeto de contenido podemos ver que contiene la estructura que cualquier otro objeto. Sin embargo, de ellos cabe destacar que la única referencia que tienen estos es la longitud del flujo de los datos de una página en bytes.

Además, encontramos un stream dentro de estos objetos que es donde realmente encontramos las instrucciones que imprimen los textos y los contenidos que tiene una página del documento con extensión PDF. Este stream comienza con la palabra clave 'BT' (Begin Text) y termina por 'ET' (End Text) lo que indica el comienzo y final de las instrucciones de una página.

```

1170 0 obj
<<
/Length 28871
>>
stream
BT
/P <</MCID 1 >>BDC
/T1_0 17.215 Tf
249.201 665.282 Td
(The)Tj
/TT0 1 Tf
27.252 0 Td
( )Tj
/T1_1 17.215 Tf
5.229 0 Td
[(axessibilit)26 (y)]TJ
/TT0 1 Tf
71.168 0 Td
( )Tj
/T1_0 17.215 Tf
5.541 0 Td
[(pac)26 (k)52 (age)]TJ
EMC
/P <</MCID 6 >>BDC
/T1_2 11.955 Tf
-198.72 -28.892 Td
(Dragan)Tj
EMC

```

*Ilustración 10: Objeto de contenido de un PDF.*

*Fuente: propia*

## Instrucciones

Las instrucciones siguen una estructura de código en la que muestran unos valores para los contenidos que van a mostrar. Si analizamos las instrucciones podemos ver que comienzan por una palabra clave como puede ser ‘/P’ que indica el comienzo de un párrafo. Sin embargo, el párrafo no tiene por qué ser de la longitud de lo que entendemos por un párrafo. Hay casos en los que cada palabra del texto es un párrafo independiente visto en estas instrucciones.

```

/P <</MCID 6 >>BDC
/T1_2 11.955 Tf
-198.72 -28.892 Td
(Dragan)Tj
EMC

```

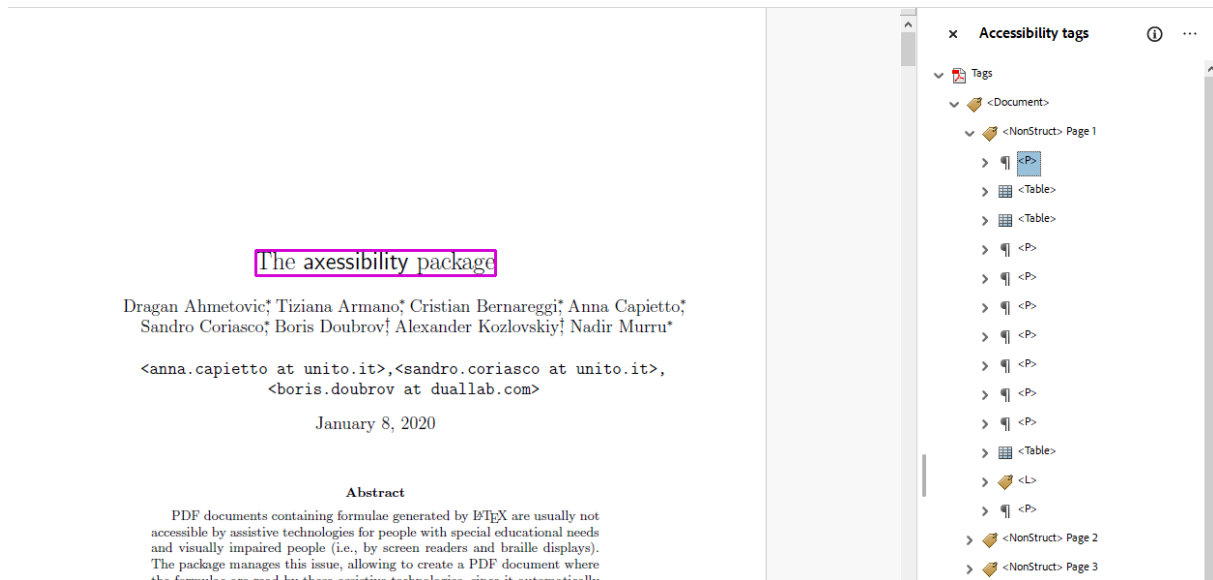
*Ilustración 11: Instrucción de imprimir un texto simple.*

*Fuente: propia*

Estos tipos de bloques son utilizados para estructurar las páginas del documento en los distintos tipos de contenido que podemos encontrarnos. Por lo tanto, las páginas se dividen a su vez en bloques de contenido que pueden contener texto.

Los bloques de contenido son utilizados posteriormente para facilitar la lectura por parte del software como pueden ser, por ejemplo, los lectores de pantalla que permiten que el usuario pueda acceder a la lectura del documento.

Esto lo podemos ver claramente en la ilustración 12, en la que se muestran los distintos bloques de contenido que tiene el documento. En él podemos ver como asocia el primer bloque de contenido con un párrafo que es el que contiene las distintas instrucciones de impresión de ese párrafo que se corresponde con el mostrado en código en la ilustración 11.



*Ilustración 12: Párrafo en PDF Debugger..*

*Fuente: propia*

Las instrucciones siguen en la misma línea por un diccionario que suele contener la palabra MCID (Marked Content ID) que hace referencia a la identificación del bloque de texto. Esta identificación es representada por un número entero que sigue a esta palabra clave. Además, esta línea acaba con la palabra clave BDC (Begin Marked Content With Property List) que indica el comienzo del bloque de contenido marcado.

Sin embargo, también pueden tener otras palabras clave como BBox en vez de MCID. Esta última (Bounding Box) hace referencia a una caja delimitadora que permite definir las coordenadas de los límites del objeto.

```
/Artifact <</BBox [365.24 230.657 368 231.713 ]/Type /Layout >>BDC
7.741 0 Td
(.)Tj
EMC
```

*Ilustración 13: Instrucción de imprimir un /Artifact.*

*Fuente: propia*

En la siguiente línea de la ilustración 11 encontramos 3 elementos. El primero hace referencia al tipo de fuente que se está estableciendo. Recordemos que esta fuente está definida en otro objeto del código. Además, tenemos un número decimal que equivale al tamaño de la fuente y está medido en puntos tipográficos (1/72 de una pulgada). Finalmente, esta línea acaba con el operador Tf (Text Font) que establece el tamaño y la fuente del contenido tomando los valores anteriores. Además, en algunos casos solo se proporciona uno de los valores de desplazamiento.



En la siguiente línea, encontramos 2 valores reales junto a un operador. Estos valores equivales al desplazamiento horizontal y al desplazamiento vertical en unidades de texto respectivamente, por lo que asignarán una nueva posición al contenido de la instrucción. Además, están seguidos por el operador Td que se utiliza para desplazar el contenido utilizando los valores anteriormente dados.

Otro operador para imprimir texto puede ser Tm (Text Matrix) que utiliza una matriz de transformación para definir las características del texto.

Cabe destacar que tanto las líneas que finalizan con el operador Tf como las que finalizan con el operador Td o Tm no siempre se encuentran en las instrucciones.

En la línea siguiente, encontramos una cadena de texto seguida del operador Tj. Este operador es el comando que se utiliza para mostrar el texto que se encuentra previo al comando. Por tanto, este operador es el que se encarga de imprimir el texto que obtiene en un documento PDF.

Sin embargo, existen otras formas de imprimir una cadena de texto. En la ilustración 14 se pueden ver notables diferencias en las que se almacenan las cadenas de texto a imprimir junto a distintos operadores para imprimir el texto.

En el primer caso, la cadena de texto se almacena entre paréntesis y el operador Tj es utilizado para mostrar estas cadenas de texto simple.

Por otro lado, el operador TJ es utilizado para mostrar texto complejo en la página. Esta muestra en el segundo caso las cadenas de texto simple haciendo pequeñas variaciones en los textos que están mostradas por los números que siguen a una cadena de texto.

El último caso es el más complejo. En este caso las cadenas de texto simple están codificadas en hexadecimal y, en vez de estar contenidas entre ( ), están contenidas entre < >. Por lo que para imprimir este texto se tiene que decodificar las cadenas hexadecimales y luego imprimir cada cadena de texto simple.

```
/P <</MCID 25 >>BDC
/T1_2 11.955 Tf
3.909 0 Td
(Capietto)Tj
EMC
```

**Ejemplo 1**

```
/P <</MCID 28 >>BDC
/T1_2 11.955 Tf
T*
(,)Tj
EMC
```

**Ejemplo 2**

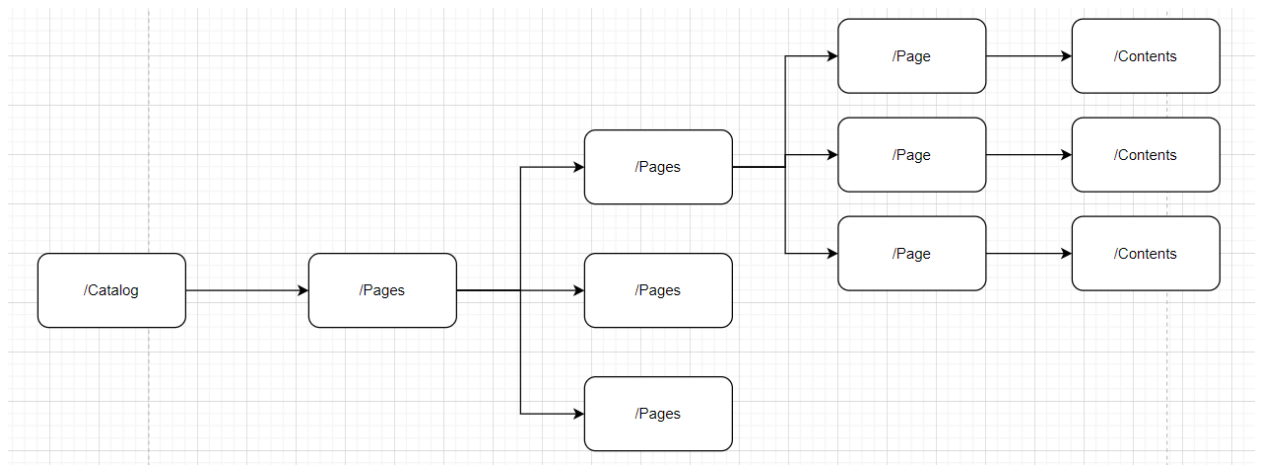
```
/P <</MCID 8 >>BDC
/T1_2 11.955 Tf
3.897 0 Td
[(Ahmeto)27.001 (vic)]TJ
EMC
```

**Ejemplo 3**

*Ilustración 14: Instrucciones de imprimir textos de formas distintas.  
Fuente: propia*

Por último, cabe destacar cómo seguimos la estructura de árbol para llegar a imprimir estos textos. Para hacernos una idea global tendremos en primer lugar un objeto de tipo catálogo que hará referencia en su diccionario a un objeto de tipo páginas. Este objeto de tipo páginas puede que tenga referencias en sus hijos a otros objetos de tipo páginas hasta que uno de estos objetos haga referencia en su diccionario a un objeto de tipo página. Cada objeto de tipo página tendrá una referencia en su diccionario al objeto de tipo contenido que tiene los contenidos de esa página además de los recursos y los tipos

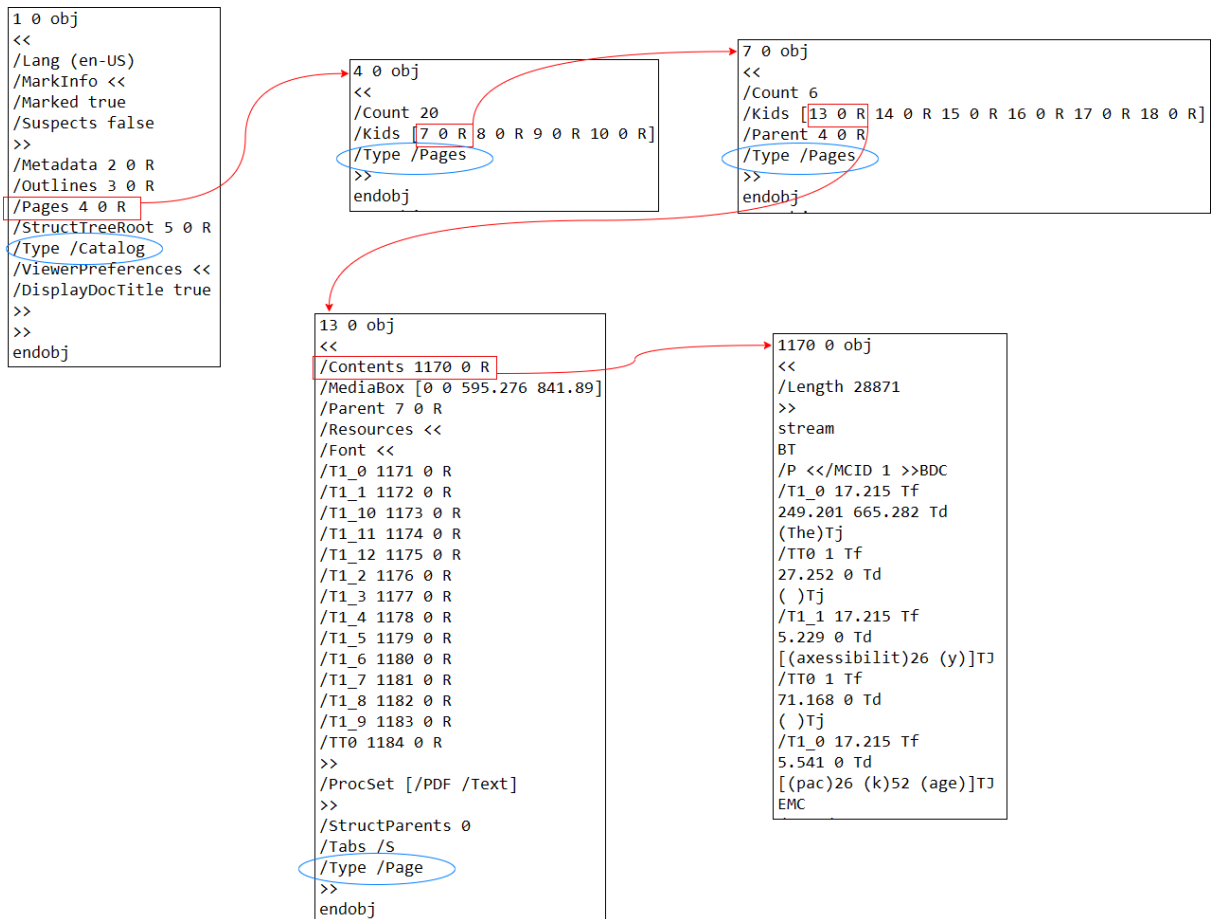
de fuentes entre otras cosas. Finalmente, este objeto de tipo contenidos tiene las instrucciones que imprimen los contenidos de las páginas como hemos visto.



*Ilustración 15: Esquema de acceso de objetos hasta los contenidos de una página.*

*Fuente: propia.*

Este esquema se puede ver perfectamente en la ilustración 15. En ella podemos ver como el primer objeto del código que es de tipo /Catalog referencia al primer objeto de tipo /Páginas que es el que tiene identificador 4. Este a su vez referencia en sus hijos al objeto con identificador 7 que también es de tipo páginas. Sin embargo, el objeto 7 referencia al objeto 13 en sus hijos, siendo este un objeto de tipo /Página. Finalmente, este objeto referencia entre otros al objeto 1170 que es el que contiene los contenidos de la página.



*Ilustración 16: Esquema de la relación de objetos.  
Fuente: propia*

## 4. Desarrollo

En el desarrollo de este proyecto identificaremos cada una de las estructuras que hemos definido anteriormente en un ejemplo de un archivo PDF y las hemos encapsulado en una estructura que hemos creado. Todo este desarrollo ha sido realizado utilizando Python como lenguaje de programación.

Cabe destacar que para el desarrollo de esta librería se ha utilizado durante todo el proceso un ejemplo de PDF que hemos definido como ‘decompressedPDF.pdf’.

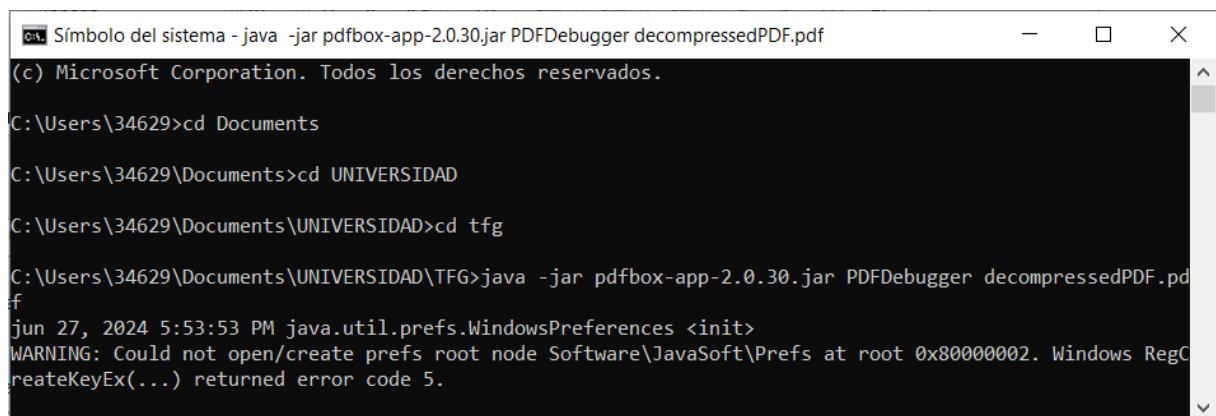
### 4.1 PDF Debugger

Entender la estructura del código del PDF es completamente necesario para poder trabajar con los contenidos que este tiene. Por ello una forma muy útil de poder visualizar la estructura del archivo desde una forma más visual era utilizar la herramienta PDF Debugger.

PDF Debugger es una herramienta diseñada específicamente para la depuración y el análisis de los archivos PDF. En ella se proporciona una forma de visualizar los contenidos de un PDF de una forma más gráfica.

Para abrir esta herramienta es necesario, en primer lugar, instalar la herramienta y desde la página de PDFBox podremos descargarnos una versión esta herramienta, en nuestro caso la versión 2.0.30.

En segundo lugar, abriremos el terminal y nos desplazaremos hasta el directorio donde tengamos el documento PDF que queremos visualizar.



```
Símbolo del sistema - java -jar pdfbox-app-2.0.30.jar PDFDebugger decompressedPDF.pdf
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\34629>cd Documents
C:\Users\34629\Documents>cd UNIVERSIDAD
C:\Users\34629\Documents\UNIVERSIDAD>cd tfg
C:\Users\34629\Documents\UNIVERSIDAD\TFG>java -jar pdfbox-app-2.0.30.jar PDFDebugger decompressedPDF.pdf
Jun 27, 2024 5:53:53 PM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegC
reateKeyEx(...) returned error code 5.
```

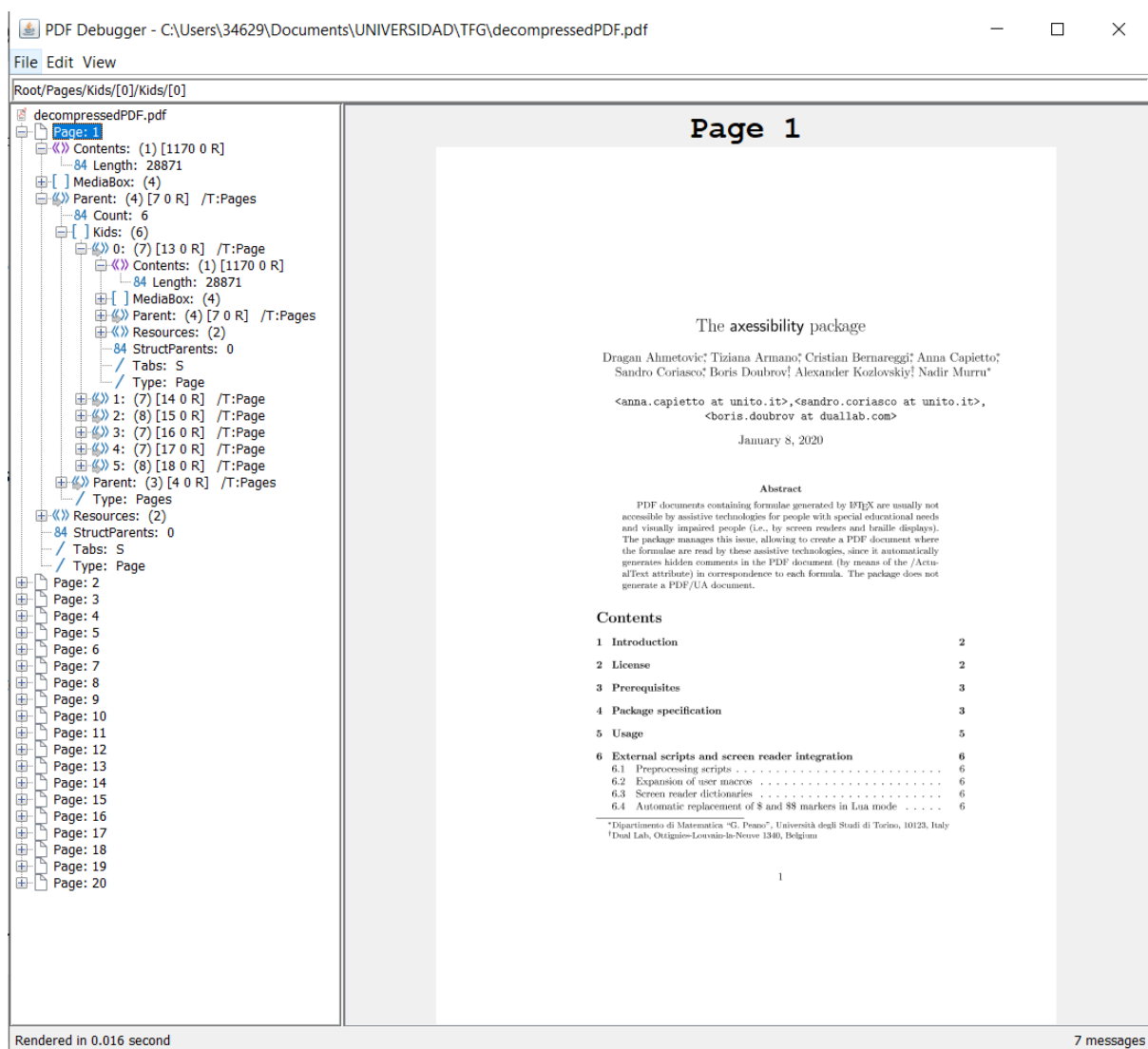
*Ilustración 17: Comando de ejecución de PDF Debugger..*

*Fuente: propia*

Finalmente, ejecutaremos el siguiente comando siendo ‘y’ y ‘z’ los dígitos que indican la versión de la herramienta que hemos instalado, 0 y 30 en nuestro caso. Además, escribiremos el nombre del documento que queremos visualizar en ‘[inputfile]’.

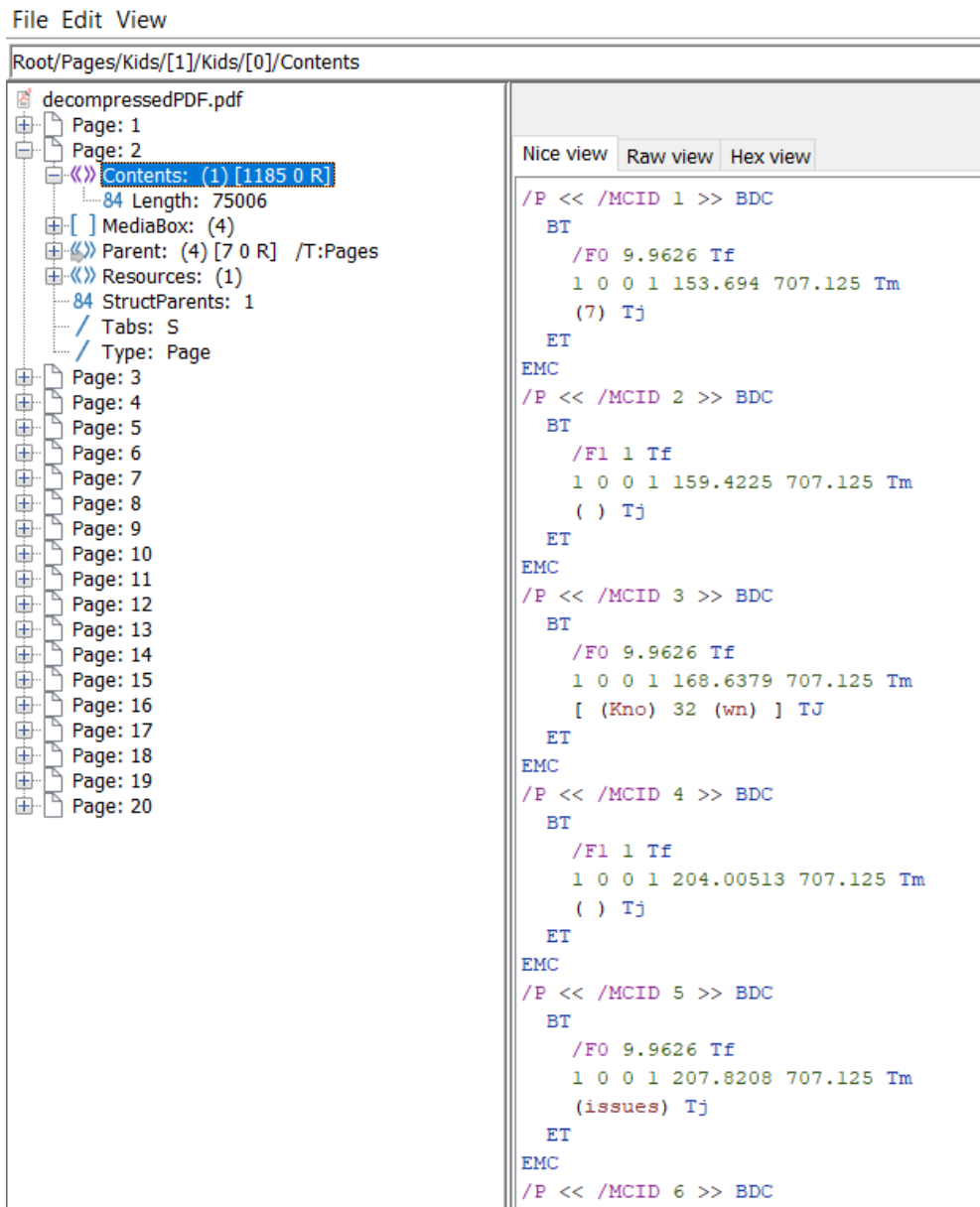
`java -jar pdfbox-app-2.y.z.jar PDFDebugger [inputfile]`

Esto nos abrirá una nueva ventana en la que podremos visualizar los contenidos del documento gráficamente.



*Ilustración 18: Vista de una hoja en PDF Debugger.  
Fuente: PDF Debugger*

Se puede seguir la estructura de árbol viendo los hijos de cada objeto y pudiendo acceder a cada una de las referencias que contiene cada objeto del PDF. Además, se muestran los contenidos de los objetos de este tipo en el que puedes ver claramente la relación entre estos contenidos en el código y en el propio documento de forma mucho más clara.



*Ilustración 19: Vista de los contenidos en PDF Debugger..  
Fuente: propia*

Por ejemplo en el caso de la ilustración 19 podemos ver los contenidos de la segunda página de forma estructurada. Podemos ver que el objeto que contiene los contenidos de esta página es el 1185 y las referencias que contiene el diccionario del objeto que referencia esta página.

Además, podemos ver que gráficamente se entiende mucho mejor la estructura de las instrucciones y comparamos con la vista cruda del código que se muestra en la ilustración 20.

```
1185 0 obj
<<
/Length 75006
>>
stream
/P <</MCID 1>>BDC
BT
/F0 9.9626 Tf
1 0 0 1 153.694 707.125 Tm
(7)Tj
ET
EMC
/P <</MCID 2>>BDC
BT
/F1 1 Tf
1 0 0 1 159.422501 707.125 Tm
( )Tj
ET
EMC
/P <</MCID 3>>BDC
BT
/F0 9.9626 Tf
1 0 0 1 168.637894 707.125 Tm
[(Kno)32(wn)]Tj
ET
EMC
/P <</MCID 4>>BDC
BT
/F1 1 Tf
1 0 0 1 204.005127 707.125 Tm
( )Tj
ET
EMC
/P <</MCID 5>>BDC
BT
/F0 9.9626 Tf
1 0 0 1 207.820801 707.125 Tm
(issues)Tj
ET
EMC
```

*Ilustración 20: Código de las instrucciones en el txt  
Fuente: propia.*

## 4.2 Estructura de encapsulación

A la hora de encapsular los contenidos de este PDF era necesario entender la estructura interna que siguen, por norma general, los documentos con esta extensión. Una vez se entiende, consideramos en encapsular la parte más extensa y complicada del archivo

que es el cuerpo del documento. Esta parte es la que más contenido tiene y donde está el mayor interés del documento.

#### 4.2.1 Expresiones regulares

Para encontrar cada elemento dentro de este código hemos hecho uso de las expresiones regulares de forma que se sigan las reglas de nomenclatura que siguen los elementos de estos archivos para poder encontrar las estructuras.

Las expresiones regulares son una forma de definir patrones en el procesamiento de los textos que facilitan la búsqueda y manipulación de estos. En ellas podemos definir las reglas que siguen algunos elementos. Por ejemplo, podemos ver que los objetos comienzan siempre por una línea que se compone por 2 números y la palabra 'obj'; y terminan por la palabra 'endobj'. Esto nos define que todo el texto que hay entre medias conforma el objeto.

Si tomamos todo el código como un archivo de texto, podemos encontrar esos patrones para poder encontrar los distintos objetos que tiene el documento, las instrucciones de imprimir texto en los objetos de contenido o cada una de las referencias del diccionario de un objeto.

De esta forma, dividimos cada elemento en otros más pequeños y vamos creando la estructura que utilizamos en el proyecto.

Para usar expresiones regulares en Python es necesario importar la librería re de Python.[13] Por ello, hemos definido algunas expresiones regulares que nos permiten encontrar los distintos elementos que hemos definido.

```
1  # EXPRESIONES REGULARES
2
3  decimalNumber = r'[\d.]+'
4  naturalNumber = r'-?'+decimalNumber
5
6  font = r'\S*'
7
8  lineFont = r'/( '+font+r') ('+decimalNumber+r') Tf\n'
9  linePos = r'(\d|\s)*?('+naturalNumber+r') ('+naturalNumber+r')
10 (Tm|Td|TD)\n'
11 lineEvery = r'T*\n'
12
13 regexID = r'(\d+) \d+ obj'
14 regexReferencia = r'\d+ \d+ R'
15 regexKids = r'/Kids \[(.*?)\]|/K \[(.*?)\]'
16 regexParent = r'/Parent (\d+) (.*?)'
17 regexContent = r'/Contents (\d+ 0 R)'
18 regexStream = r'stream(.*?)endstream'
19
20 regexObject = r'\d+ \d+ obj(?:\n<<[\s\S]*?endobj|\nendobj)'
21
22 regexContentReference =
23 r'((?!<|>|[\s\S]) (.*?)\n|(<<\n.*?>>)\n|([\s\S])\n(/>>))'
24 regexReference = r'/(\S+) '+regexContentReference
```



```

25 regexRealContent = r'<<(.*?)>>\nendobj '
26
27 regexTexts = r'(/Span|/P|/LBody|/Artifact) (.*?)EMC'
28
29 regexTextBegin1 = r'/MCID (\d+) (.*?) (Tj|TJ) '
30 regexTextBegin2 = r'/BBox(.*?) (Tj|TJ) '
31
32 # Expresiones regulares para las líneas de impresión del texto
33 regexTextContent1 =
34 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\((.*?)\)Tj'
35 regexTextContent2 =
36 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[(((?!\\)\(.*?))
37 (?!\\)\])?(-?(\d|s)*(?!\\)\(.*?(?!\\)\))*\]TJ'
38 regexTextContent3 =
39 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[?(<(.*?)>(-
40 (\d|s)*<.*?>)*\]?T(J|j)'
41 regexTextContent4 =
42 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[-
43 ?(\d)\(\w\) \]TJ'

```

Entre estas expresiones regulares debemos destacar algunas:

- **regexObject** es la expresión regular de un objeto en la que comprobamos que comiencen por un número que se corresponde con el id del objeto, seguido de otro número que sería la versión de generación y la palabra clave 'obj'; y terminen por la palabra clave 'endobj'.
- **regexID** es la expresión regular que recoge la primera línea de un objeto en la que se encuentra su número de identificación.
- **regexKids** es la expresión regular que obtiene la lista de referencias de los hijos de un objeto dado.
- **regexTexts** es la expresión regular que recoge todo el contenido de una instrucción de impresión de textos.
- **lineFont** es la expresión regular que recoge la línea en la que definen el tipo de fuente a utilizar y el tamaño del texto.
- **linePos** es la expresión regular que recoge la línea en la que se definen el desplazamiento horizontal y vertical de un texto.
- **regexTextContent1, 2, 3 y 4** son las expresiones regulares que recogen los textos que se van a imprimir de las distintas formas que pueden estar agrupadas las cadenas de texto: cadenas simples, listas de cadenas, listas de cadenas hexadecimales...

#### 4.2.2 Estructuras

En esta sección se presentan las estructuras que se han creado para poder simular y encapsular correctamente los elementos que hemos definido de la estructura interna del PDF previamente convertido en txt.

#### 4.2.2.1 Clases

La forma principal en la que encapsulamos los contenidos que se encuentran en el cuerpo del archivo PDF es en objetos. Para ello hemos creado una clase objeto en la que podemos obtener su número de identificación y su contenido, es decir, todo el texto que aborda el objeto en el código desde su identificación hasta la palabra clave 'endobj'. Además, también recogeremos la lista de objetos hijos del objeto si tiene, las referencias a otros objetos en la clave '/Contents', las referencias y, en caso de ser un objeto de tipo contenido, una lista de los textos que se imprimen para una página.

```
1 # Clase de objeto del cuerpo del PDF
2 class Object:
3     def __init__(self, content, id):
4         self.id = id
5         self.content = content
6         self.kids = []
7         self.contents = ""
8         self.printedTexts = []
9         self.references = {}
10
11
12 # Clase de la instrucción de imprimir un texto de un PDF
13 class PrintedText:
14     def __init__(self, id, font, size, rowText, text, posX, posY):
15         self.id = id
16         self.font = font
17         self.size = size
18         self.rowText = rowText
19         self.text = text
20         self.posX = posX
21         self.posY = posY
```

En segundo lugar, es necesario crear la estructura que almacena los textos que se imprimen en la clase PrintedText. De estos textos almacenamos el número de identificación, el tipo de fuente, el tamaño del texto, el desplazamiento horizontal y el desplazamiento vertical. Además, se almacena el texto en crudo, es decir, la instrucción de imprimir un texto conteniendo la cadena de texto en la forma de esa instrucción (cadena simple, lista de cadenas simples o lista de cadenas hexadecimales) y el propio texto que se imprime en el que se unen las cadenas simples para generar el texto que luego se visualiza al abrir el documento.

#### 4.2.2.2 Listas

Por otra parte, vamos a tener una lista de todos los objetos que tiene el cuerpo del documento PDF y una lista con los objetos que son de tipo contenido que nos permitan acceder a todos los objetos que existen. Desde estas listas podremos acceder a las referencias y contenidos que almacena el código del archivo.

```
1 # Cargar objetos
2 listObjects = []
3 listContents = []
```

### 4.3 Carga de los objetos

Para obtener el código de un documento PDF basta con cambiar la extensión de ese mismo documento a txt. De esta forma podremos ver el contenido en crudo del archivo y donde encontraremos todas las estructuras que hemos mencionado anteriormente.

Posteriormente abriremos el archivo en formato de lectura para acceder a los contenidos que están en el archivo txt pero que representan un PDF.

```
1 # Abrir el archivo PDF
2 file_path =
3 "c:/Users/34629/Documents/UNIVERSIDAD/TFG/decompressedPDF.txt"
4 try:
5     with open(file_path, 'r', encoding='utf-8', errors='ignore') as
6 archivo:
7         contenido = archivo.read()
8 except UnicodeDecodeError as e:
9     print("Error de decodificación Unicode:", e)
```

Para cargar los objetos utilizaremos las expresiones regulares explicadas anteriormente en la lista listObjects. En primer lugar, se cargan todos los textos que coinciden con la expresión regular regexObjects y se crearán las instancias de la clase objetos pasándoles el identificador del objeto y el contenido del objeto, es decir, el propio texto del objeto, como atributos de la clase.

En segundo lugar, teniendo el contenido de cada objeto, tratamos de cargar le resto de atributos. Intentamos obtener los identificadores de los hijos y los identificadores de los objetos de tipo contenido en las referencias '/Kids' o '/K' y '/Contents' respectivamente. Además, cargaremos las referencias del diccionario de los objetos de cada objeto.

En tercer lugar, por cada objeto de contenido obtenido, incluiremos el objeto en la lista listContents. Además, por cada hijo de un objeto obtenido, incluiremos el objeto en la lista de objetos de ese objeto padre.

```
1 objetos = re.findall(regexObject, contenido)
2
3
4 for objeto in objetos:
5     id = getID(objeto)
6     newObject = Object(objeto, id)
7     listObjects.append(newObject)
8
9 # Cargar objetos
10 for obj in listObjects:
11     children = getKids(obj.content) # Cargar ids de los hijos
12
13     obj.contents = getContents(obj.content)
14     if obj.contents != None:
15         listIntContents.append(obj.contents)
16
17
18 # Cargar referencias
```

```

19     obj.references = getReferences(obj.content)
20
21     # Cargar la la lista de objetos hijos de un objeto
22     for obj2 in listObjects:
23         for child in children:
24             if obj2.id == child:
25                 obj.kids.append(obj2)
26
27     # Cargar la lista de ids de los objetos de tipo contenido
28     for n in listIntContents:
29         if obj.id == n:
30             listContents.append(obj)
31
32
33 # Cargar los textos de los objetos de tipo contenido
34 for objContent in listContents:
35     lista = re.findall(regexTexts,objContent.content,re.DOTALL)
36     for text in lista:
37         #obj2.printedTexts.append(parse_text(text[1]))
38         parse_text(objContent,text[1])

```

Finalmente, por cada objeto de contenido obtendremos mediante el uso de las expresiones regulares las instrucciones de los textos que se imprimen en cada página. Para ello se hace uso de la función `parse_text()` que carga los valores de la clase `PrintedTexts` para cada texto.

De esta forma encapsulamos todos los objetos del documento y, para cada objeto de tipo contenido, encapsulamos todos los textos que se imprimen con sus características principales.

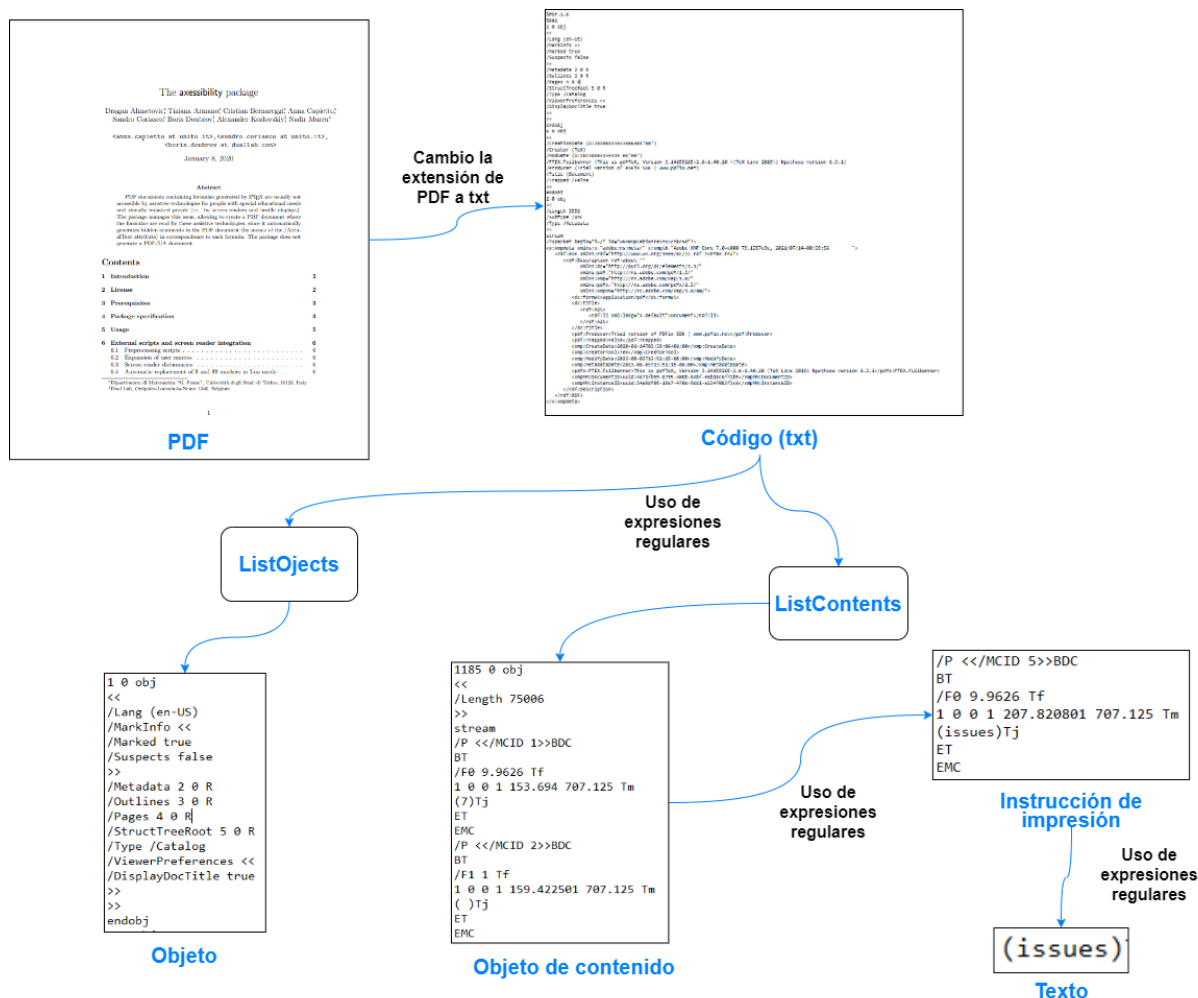


Ilustración 21: Esquema de los objetos que se encapsulan,  
Fuente: propia.

La ilustración 21 muestra un esquema de la forma en la que cargamos la estructura que hemos creado. En ella cambiamos, en primer lugar, la extensión del PDF a txt. De esta forma obtenemos el código que representa el documento.

En segundo lugar, utilizamos las expresiones regulares que hemos definido para obtener los objetos que guardamos en la lista listObjects. Además, los objetos de contenido los almacenamos en otra lista listContents.

En tercer lugar, volvemos a usar las expresiones regulares para obtener las instrucciones que representan la impresión del texto en el documento y poder almacenar los textos que finalmente se imprimen.

## 4.4 Diseño funcional

En esta sección se explicará cómo funciona la herramienta desarrollada al ejecutarla y cómo utilizarla.

### 4.4.1 Ejecución del programa

Para la ejecución del programa se abrirá un entorno de ejecución en el que se pueda ejecutar la solución diseñada. En este caso, el entorno utilizado ha sido Visual Studio

Code[9]. Una vez abierto seleccionaremos el archivo que contiene el código del programa creado y lo ejecutaremos.

```
PS C:\Users\34629> & C:/Users/34629/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/34629/Documents/UNIVERSIDAD/TFG/CodigoFuenteTFG.py
```

*Ilustración 22: Terminal al ejecutar la herramienta desarrollada.  
Fuente: propia.*

#### 4.4.2 Menú

Una vez tenemos cargados todos los contenidos en las estructuras que hemos definido, hemos llevado a cabo algunas acciones en las cuales podemos comprobar cómo está estructurado el documento PDF y como hemos conseguido encapsular esos contenidos.

Para ello, hemos creado un menú en el que tendremos varias opciones que se corresponde cada una con una acción descrita en el menú. Las acciones del menú están descritas en la ilustración 23.

```
-----
Menú de opciones
1. Imprimir el árbol de estructura del PDF
2. Imprimir una página del PDF
3. Imprimir el PDF entero
4. Imprimir un texto por id
5. Imprimir referencias de un objeto
6. Modificar objeto
7. Descargar PDF
8. Gestionar los hijos de un objeto
9. Salir
-----
Selecciona una opción (1-9):
```

*Ilustración 23: Menú de opciones de la plataforma desarrollada.  
Fuente: propia.*

Vamos a describir cada una de las opciones del menú, explicando cómo se realiza cada funcionalidad de la acción.

#### 4.4.3 Opción 1

La opción 1 nos da la opción de imprimir el árbol de estructura del PDF. Con esta opción, imprimimos los identificadores de todos los objetos que están relacionados en la ruta que llega hasta los objetos de tipo contenido de las páginas del PDF.

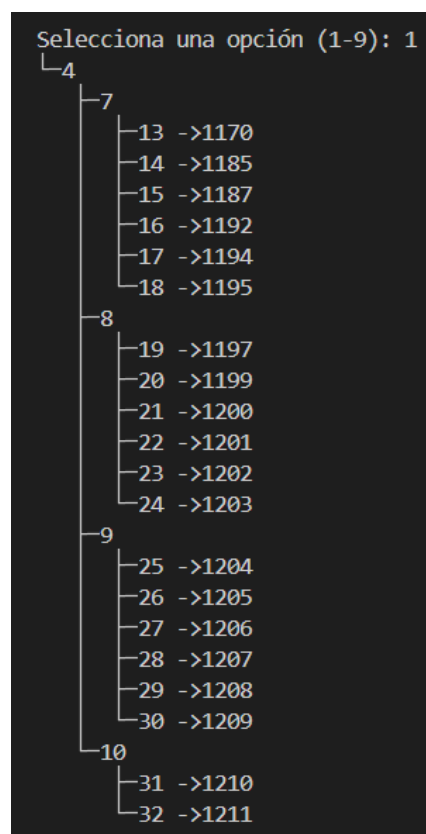
En esta opción seleccionamos el primer objeto de tipo páginas que es el objeto con número de identificación 4. A partir de este seleccionaremos sus hijos que son de tipo páginas hasta llegar a los hijos que son de tipo página. Por último, seleccionamos los objetos de tipo contenido que se encuentran referenciado en los objetos de tipo página.

```

1  # Método que imprime el árbol de estructura desde los objetos de tipo
2  # página hasta los objetos de contenido de las páginas
3  def print_struct_tree(nodo, prefijo='', es_ultimo=True):
4      print(prefijo, end='')
5      print('└─' if es_ultimo else '├─', end='')
6      print(nodo.id, end='')
7      print(f' ->{nodo.contents}' if (nodo.contents != None) else '')
8
9      prefijo += '    ' if es_ultimo else '│  '
10     cantidad_hijos = len(nodo.kids)
11
12     for i, hijo in enumerate(nodo.kids):
13         es_ultimo = i == cantidad_hijos - 1
14         print_struct_tree(hijo, prefijo, es_ultimo)

```

En el ejemplo podemos ver que los objetos 4, 7, 8, 9 y 10 son de tipo páginas. Estos hacen referencia a los objetos de tipo página (13, 14, 15, 16, 17 y 18 para el objeto 7) y que ellos tienen la referencia del objeto de tipo contenido de la página (1170 para el objeto 13).



*Ilustración 24: Árbol de estructura hasta los contenidos de las páginas.  
Fuente: Plataforma desarrollada*

#### 4.4.4 Opción 2

La opción 2 nos permite imprimir una página del documento. Una vez hemos encapsulado todos los textos y contenidos que hay en cada página basta con imprimir todos los textos de una página. Puesto que están ordenados los objetos de tipo

contenido en listContents se pide el número de la página que se quiere imprimir y ese número indica la posición en la lista de la página.

Puesto que ya hemos encapsulado cada uno de los textos de una página junto a las características del texto que están definidas en la instrucción, accederemos a cada uno de esos textos que se encuentran en el objeto de contenido de esa página para imprimir la página entera.

Además, para imprimir los textos, comprobamos si, en el caso de que exista desplazamiento en el eje vertical, haya una diferencia notable entre los desplazamientos de ambos textos y en ese caso se entiende como un salto de línea.

```
1  # Método que imprime los textos de una de las páginas
2  def print_sheet(obj):
3      previous_height = obj.printedTexts[0].posY
4      for text in obj.printedTexts:
5          if(text.posY != ''):
6              if (abs(float(text.posY) - float(previous_height)) > 10
7 and text.posY != '0'):
8                  print('\n'+text.text, end='')
9              else:
10                 print(text.text, end='')
11                 previous_height = text.posY
12             else:
13                 print(text.text, end='')
```

De esta forma obtenemos todos los textos por separado impreso. Es necesario destacar, además, que, al estar predefinidas las posiciones de los textos, hay veces que los espacios forman parte de los textos, sin embargo, hay veces que estos textos no tienen instrucciones que imprimen espacios en blanco entre ellos.



```

Selecciona una opción (1-9): 2
Por favor, escribe la página que quieres imprimir (o '0' para volver al menú principal): 2
7 Known issues7
8 Implementation7
9 History20
1 Introduction
This package focuses on the specific problem of the accessibility of PDF documents
generated by L A T E X for visually impaired people and people with special educa-
tional needs. When a PDF document is generated starting from L A T E X, formulae
are not accessible by screen readers and braille displays. They can be made acces-
sible by inserting a hidden comment, i.e., an ActualText, similarly to the case of
web pages. This can be made, e.g., by using the L A T E X package pdfcomment.sty.
In any case, this task must be manually performed by the author and it is surely
inefficient, since the author should write the formulae and, in addition, insert a
description for each formula. Note also that the package pdfcomment.sty does not
allow to insert special characters like `backslash', `brace', etc, in the comment.
Moreover, with these solutions, the reading is bothered since the screen reader
first reads incorrectly the formula and then, only as a second step, provides the
correct comment of the formula. There are also some L A T E X packages that try to
improve the accessibility of PDF documents produced by L A T E X. In particular,
the packages accsupp.sty, accessibility
meta.sty and tagpdf have been developed
in order to obtain tagged PDF documents. The package accsupp.sty develops
some interesting tools for commenting formulae using also special characters \ (pos-
sibility that is not available, e.g., in the pdfcomment.sty package). The package
tagpdf widely further developed tagging functionalities, along the most recent
specifications for PDF documents accessibility. However, all of the above are not
automatized methods, since the comment and tags must be manually inserted by
the author. The package accessibility
meta.sty is an improved version of the pack-
age accessibility.sty. This package allows the possibility of inserting several tags
for sections, links, figures and tables. However, even if these tags are recognized
by the tool for checking tags of Acrobat Reader Pro, they are not always recog-
nized by the screen readers. Moreover, this package does not manage formulae.
Our package automatically produces an ActualText corresponding to the L A T E X
commands that generate the formulae. This ActualText is hidden in the PDF
document, but the screen reader reads it without reading any incorrect sequence
before. Additional functionalities, implemented in this version, are available when
the typeset is done by means of luaL A T E X (see below).

```

*Ilustración 25: Página impresa del PDF.  
Fuente: Plataforma desarrollada*

#### 4.4.5 Opción 3

La opción 3 ejecutará la acción de imprimir el PDF entero. Esta vez, accedemos a la lista `listContents` y por cada objeto, puesto que ya están ordenados en esta lista, imprimiremos los textos de la misma forma que la opción anterior.

```

the formulae are read by these assistive technologies, since it automatically
generates hidden comments in the PDF document \ (by means of the /ActualText
attribute\ ) in correspondence to each formula. The package does not
generate a PDF/UA document.
Contents
1Introduction2
2License2
3Prerequisites3
4Package specification3
5Usage5
6External scripts and screen reader integration6
6.1Preprocessing scripts.....6
6.2Expansion of user macros.....6
6.3Screen reader dictionaries.....6
6.4Automatic replacement of $
and $$ markers in Lua mode.....6
* Dipartimento di Matematica \G. Peano", Universit  degli Studi di
Torino, 10123, Italy Dual Lab, Ottignies-Louvain-la-Neuve 1340, Belgium
1

7 Known issues7
8 Implementation7
9 History20
1 Introduction
This package focuses on the specific problem of the accessibility of PDF documents
generated by L A T E X for visually impaired people and people with special educa-
tional needs. When a PDF document is generated starting from L A T E X, formulae
are not accessible by screen readers and braille displays. They can be made acces-
sible by inserting a hidden comment, i.e., an ActualText, similarly to the case of
web pages. This can be made, e.g., by using the L A T E X package pdfcomment.sty.
In any case, this task must be manually performed by the author and it is surely
inefficient, since the author should write the formulae and, in addition, insert a
description for each formula. Note also that the package pdfcomment.sty does not
allow to insert special characters like `backslash', `brace', etc, in the comment.
Moreover, with these solutions, the reading is bothered since the screen reader
first reads incorrectly the formula and then, only as a second step, provides the
correct comment of the formula. There are also some L A T E X packages that try to
improve the accessibility of PDF documents produced by L A T E X. In particular,
the packages accsupp.sty, accessibility
meta.sty and tagpdf have been developed

```

*Ilustraci n 26: PDF completo impreso..  
Fuente: propia.*

#### 4.4.6 Opci n 4

La opci n 4 nos permite imprimir un texto por el id de la instrucci n que contiene el texto. Para ello hay que tener en cuenta que al encapsular los textos hay algunos, como los de tipo ‘/Artifact’, que no tienen un identificador. Para estos casos, les hemos asignado el identificador 0. Adem s, algunas instrucciones contienen varios textos a imprimir, por lo que a todos esos textos les asignamos el mismo identificador.

Esta opci n ayuda mucho a entender como est n encapsulados los contenidos del documento PDF. Para imprimir un texto tendremos que indicar, en primer lugar, la p gina del texto que queremos imprimir. Con esto accederemos al objeto de tipo contenidos que contiene los textos de esa p gina.

Posteriormente, imprimimos todos los identificadores de los textos que hemos encapsulado del objeto de tipo contenido de esa página y solicitamos al usuario que elija un identificador para imprimir.

```
1 # Método que imprime los textos de un objeto cuyo id es uno pasado
2 como argumento
3 def print_text_by_id(id, obj):
4     print()
5     print(f"El id es {id}")
6     for text in obj.printedTexts:
7         if text.id == id:
8             print(f'Tipo de fuente:\{text.font}\, Tamaño:\{text.size}\, Pos
9 horizontal:\{text.posX}\, Pos Vertical:\{text.posY}\, Texto:\{text.text}\')
```

Una vez el usuario selecciona un identificador del texto que quiere imprimir, se imprime el texto con sus características obtenidas de la instrucción de imprimir el texto.

```
[0], [414], [414], [414], [414], [414], [415], [415], [415], [415]
[415], [0], [0], [0], [416], [416], [416], [416], [416], [417]
[418], [419], [420], [421], [422], [423], [424], [425], [426], [427]
[428], [429], [430],
Selecciona un id para imprimir: 404

El id es 404
Tipo de fuente:'F2', Tamaño:'9.9626', Pos horizontal:'456.624786', Pos Vertical:'301.673004', Texto:'\old\')
```

*Ilustración 27: Ejemplo de imprimir un texto de una página.*

*Fuente: Plataforma desarrollada*

#### 4.4.7 Opción 5

La opción número 5 nos permite imprimir las referencias contenidas en el diccionario que tienen los objetos en su estructura. Recordemos que este diccionario está compuesto por duplas clave-valor en las que la clave comienza por el carácter '/' y posteriormente le sigue un valor.

```
1 # Método que imprime las referencias del diccionario de un objeto
2 def print_referencias(obj):
3     print()
4     for clave, valor in obj.references.items():
5         print(f"{clave, valor}")
6     print()
```

Una vez hemos encapsulado los objetos, para acceder a las referencias de uno de ellos solicitamos al usuario que nos indique el identificador del objeto del que quiere ver sus referencias. Tras indicarlo, se imprime el diccionario separando la clave del valor para que se puedan diferenciar sin problema.

```
Selecciona una opción (1-9): 5
Por favor, escribe el id del objeto. Debe ser un número del 1 al 2933 (o '0' para volver al menú principal): 45

('K', '[729 0 R 730 0 R 731 0 R 1252 0 R 790 0 R]')
('S', '/NonStruct')
('T', '(Page 13)')
```

*Ilustración 28: Ejemplo de diccionario de referencias de un objeto.*

*Fuente: Plataforma desarrollada*

#### 4.4.8 Opción 6

La opción 6 trata de investigar cómo podríamos modificar el documento una vez lo tenemos encapsulado. Para ello permitimos 2 opciones: modificar las referencias de un objeto o modificar los textos de una página.

Para modificar una referencia de un objeto necesitamos que el usuario indique el identificador del objeto del cual quiere cambiar sus referencias. Una vez indicado se imprime el diccionario que contiene el objeto de la misma forma que en la opción anterior y se le pide al usuario que indique la clave de la referencia que desea modificar.

Una vez el usuario ha indicado la clave, se pide un nuevo valor para esta clave. Puesto que no hay restricciones a la hora de cambiar el valor de la referencia hay que tener en cuenta de que si se intentara volver a generar el PDF con ese código podría corromperse el archivo. Por ejemplo, si una instancia del diccionario tiene una referencia a un objeto y se cambia por cualquier texto.

```
1  # Método que modifica el valor de una referencia de un objeto
2  def modificar_referencia(obj):
3      while True:
4          print_referencias(obj)
5
6          clave = input("Escriba la clave del valor que quiere
7  cambiar: ")
8          if clave in obj.references:
9              antiguo_valor = obj.references[clave]
10             print(f"\n[{clave} -> {obj.references[clave]}\n")
11             nuevo_valor = input("Escriba el nuevo valor de esta
12 referencia. Ten en cuenta de que puedes estropear el PDF (0 para
13 volver): ")
14             if nuevo_valor != '0':
15
16                 obj.references[clave] = nuevo_valor
17
18                 listCambiosReferencias.append([obj, clave,
19 antiguo_valor, nuevo_valor])
20                 print("Valor cambiado...")
21                 break
22             else:
23                 break
24         else:
25             print(f"La clave '{clave}' no existe en el
26 diccionario.")
```

Por otra parte, para modificar un texto de un objeto se pedirá al usuario que indique el número de la página a la que pertenece el texto que se desea cambiar. Posteriormente, se imprimen todos los identificadores de los textos que están en el objeto de tipo contenido de la página seleccionada de la misma forma que se hace en la opción 4.

Además, al igual que en la opción 4 se imprimen las características del texto y se pide un nuevo valor para el texto seleccionado. Este nuevo texto será el nuevo valor del texto. Hay que tener en cuenta de que los textos tienen un espacio para ser impresos que está definido en las características. Si se intenta volver a generar el PDF con esta

estructura, hay que tener en cuenta de que se pueden superponer los textos o incluso corromper la página.

En caso de haber varios textos con el mismo identificador, se pedirán nuevos valores para cada uno de los textos que tengan ese identificador en este objeto.

```
1 # Método que modifica el valor de un texto de un objeto
2 def modificar_texto(id_text, obj):
3     for text in obj.printedTexts:
4         if text.id == id_text:
5             nuevo_valor = input("Inserte el nuevo valor de este
6 texto: ")
7             antiguo_valor = text.rowText
8             text.text = nuevo_valor
9
10            listCambiosTexts.append([obj, id_text, antiguo_valor,
11 nuevo_valor])
12            print("Valor cambiado...")
```

Finalmente, estos cambios realizados se almacenan. Los cambios de referencias de objetos se almacenan en una lista mientras que los cambios de textos de las páginas se almacenan en otra lista distinta.

#### 4.4.9 Opción 7

La opción 7 nos permite descargarnos el código del PDF de nuevo. Sin embargo, el interés de esta opción está en que el documento que nos descargamos tiene los cambios aplicados anteriormente en la estructura que hemos creado. Como hemos explicado anteriormente, hemos aplicado 2 cambios: las referencias del diccionario de los objetos y los textos de las páginas.

Para modificar en el propio documento, tenemos que acceder a la lista que almacena los cambios de las referencias. Por cada cambio se habrá almacenado una lista con el identificador del objeto que hemos cambiado, la clave de la referencia, el valor que tenía antes esa instancia del diccionario y el nuevo valor que hemos asignado.

Una vez tenemos los cambios encapsulados, por cada cambio en las referencias de un objeto, accedemos al documento y, mediante expresiones regulares, encontraremos el objeto que queremos modificar. Posteriormente, encontraremos la referencia que queremos cambiar con su antiguo valor y lo cambiaremos por el nuevo valor asignado.

```
1 # Método que modifica las referencias de un objeto en el contenido
2 de código del documento PDF pasado
3 def modificar_pdf_referencias(content, cambio):
4     regexObj = str(cambio[0].id)+r" 0 obj.*?endobj"
5
6     match = re.search(regexObj, content, re.DOTALL)
7     if match:
8         last_match = match.group()
9         regexReference = r'/' + str(cambio[1]) + r'
10 '+regexContentReference
11         match1 = re.search(regexReference, match.group())
12         if match1:
```

```

13         last_match1 = match1.group()
14         new_match1 = match1.group().replace(cambio[2],cambio[3])
15
16         new_match = match.group().replace(last_match1,new_match1)
17
18         new_content = content.replace(last_match,new_match)
19
20     return new_content

```

De la misma forma, para realizar los cambios de los textos de las páginas almacenamos en una lista. En esta por cada modificación aplicada añadiremos una lista que guarde el identificador del objeto de tipo contenido que hemos modificado, el identificador del texto, el antiguo valor del texto y el nuevo valor del texto que hemos asignado.

De esta forma, encontramos a través de las expresiones regulares el objeto que hemos modificado. Posteriormente, encontramos el texto con el identificador que hemos cambiado con el valor antiguo del texto y lo cambiamos por el nuevo valor de forma similar al caso anterior.

```

1  # Método que modifica los textos de un objeto en el contenido de
2  código del documento PDF pasado
3  def modificar_pdf_textos(content, cambio):
4      regexObj = str(cambio[0].id)+r" 0 obj.*?endobj"
5      new_content = content
6      match = re.search(regexObj,content,re.DOTALL)
7      if match:
8          last_match = match.group()
9          regexReference = r'MCID '+str(cambio[1])+r'.*?EMC'
10         match1 = re.search(regexReference, match.group(), re.DOTALL)
11         if match1:
12             last_match1 = match1.group()
13             textSearch = cambio[2].replace("(", "\(")
14             textSearch2 = textSearch.replace(")", "\)")
15             regexRowText = r'(\('+textSearch2+r'\))(T(j|J))'
16
17             match2 = re.search(regexRowText,match1.group())
18             if match2:
19                 last_match2 = match2.group()
20                 newText = "("+cambio[3]+"Tj"
21                 new_match1 =
22 match1.group().replace(last_match2,newText, 1)
23
24                 new_match =
25 match.group().replace(last_match1,new_match1)
26
27                 new_content = content.replace(last_match,new_match)
28             else:
29                 regexRowText2 = r'(\['+textSearch2+r'\])(T(j|J))'
30                 match3 = re.search(regexRowText2,match1.group())
31                 if match3:
32                     last_match3 = match3.group()
33                     newText = "["+cambio[3]+"]TJ"
34                     new_match1 =
35 match1.group().replace(last_match3,newText, 1)
36

```

```
37             new_match =
38 match.group().replace(last_match1,new_match1)
39
40             new_content =
41 content.replace(last_match,new_match)
42     return new_content
```

Hay que tener en cuenta de que estas modificaciones pueden suponer una alteración que acabe por corromper un texto, una página o incluso el archivo entero.

Por último, nos descargamos un nuevo archivo txt que contiene las modificaciones que hemos realizado en la ejecución del programa. De forma inversa a como lo hicimos al principio, cambiaremos la extensión del archivo txt a PDF para poder ver su contenido de forma habitual.

## The **axessibility** package

Dragan Ahmetovic\*, Tiziana Armano\*, Cristian Bernareggi\*, Anna Capietto\*,  
Sandro Coriasco\*, Boris Doubrov†, Alexander Kozlovskiy†, Nadir Murru\*

<anna.capietto at unito.it>,<sandro.coriasco at unito.it>,  
<boris.doubrov at duallab.com>

January 8, 2020

### **Abstract**

*Ilustración 29: Primera página del PDF utilizado.  
Fuente: propia*

## he modificado pdf

Dragan Ahmetovic, Tiziana Armano, Cristian Bernareggi, Anna Capietto,  
Sandro Coriasco, Boris Doubrov, Alexander Kozlovskiy, Nadir Murru

<anna.capietto at unito.it>, <sandro.coriasco at unito.it>,  
<boris.doubrov at duallab.com>

January 8, 2020

### Abstract

*Ilustración 30: Primera página modificada del PDF utilizado.  
Fuente: propia*

#### 4.4.10 Opción 8

La opción 8 nos permite acceder a los hijos de un objeto. Además, permite eliminar o añadir nuevos hijos a un objeto, modificando así la estructura. Esta opción permite navegar por el árbol de objetos accediendo a sus hijos.

Para ello, hemos encapsulado los hijos de cada objeto en una lista con los objetos hijos. No encapsulamos el identificador del objeto, si no el propio objeto en sí.

En primer lugar, se solicita el identificador del objeto que se quiere visualizar y se muestran sus hijos. Posteriormente, se dará opción de añadir o eliminar un hijo, o simplemente acceder a los hijos de uno de estos objetos permitiendo seguir navegando por el árbol de objetos.

```
1 # Método que gestiona si se quiere añadir, eliminar o ver los hijos
2 de un objeto
3 def gestionar_hijos(id):
4     kids = print_kids(listObjects[id-1])
5     if kids == []:
6         return
7     else:
8         show_kids_menu()
9         opción = input("Elija una opción: ")
10        if opción == '0':
11            return
12        elif opción == '1':
13            new_id = input("Elija un hijo: ")
14            if any(kid.id == int(new_id) for kid in kids):
15                delete_kid(listObjects[id-1], int(new_id))
16            else:
17                print("El id no es no hijo de este objeto")
18            return
```



```
19     elif opcion == '2':
20         new_id = input("Elija un hijo: ")
21         if any(int(new_id) < 2933 and int(new_id) > 0):
22             add_kid(listObjects[id-1], int(new_id))
23         else:
24             print("El id no es válido")
25             return
26     elif opcion == '3':
27         new_id = input("Elija un hijo: ")
28         if any(kid.id == int(new_id) for kid in kids):
29             gestionar_hijos(int(new_id))
30         else:
31             print("El id no es no hijo de este objeto")
32             return
```

## 5. Conclusiones y trabajo futuro

Una vez hemos desarrollado el proyecto y cumplidos los objetivos propuestos, podemos concluir que se ha obtenido una estructura que es capaz de encapsular los contenidos del PDF. Además, esa estructura se puede visualizar fácilmente ayudando a la comprensión de esta estructura interna.

Por otra parte, podemos ver que la estructura desarrollada se corresponde con la de archivos PDF. Se distinguen los distintos objetos que contiene la estructura, se permite visualizar sus referencias, tipos e incluso se permite navegar por el árbol de objetos que es el que enlaza unos objetos con otros creando así verdaderamente la estructura.

También podemos ver, de los objetos de tipo contenido, las distintas formas e instrucciones que existen para imprimir los textos que posteriormente podemos visualizar en el documento. De ellos además obtenemos la forma en la que esto se posicionan, sus fuentes, su tamaño... permitiendo de esta forma almacenar todos los contenidos que tienen las páginas del documento.

De cara al futuro, se pueden plantear algunas mejoras o nuevas implementaciones que permitirían incrementar el valor del trabajo ya realizado. Una de estas mejoras es la de obtener las imágenes que se visualizan en los documentos y como se definen sus tamaños, así como poder obtener las imágenes dado su stream de datos.

Otra posible mejora, sería la diseñar una nueva implementación que permita imprimir los textos y las imágenes obtenidas en un nuevo archivo conociendo su desplazamiento en el eje horizontal y vertical. Esto intentaría recrear de forma fiel la forma en la que estos contenidos son impresos posteriormente en las distintas páginas.

# Bibliografía

- [1] «Adobe». [En línea]. Disponible en: <https://www.adobe.com/>
- [2] Adobe Systems Incorporated, *ISO32000*.
- [3] D. Jhonson, «PDF`s popularity online». [En línea]. Disponible en: <https://pdfa.org/pdfs-popularity-online/>
- [4] «Python». [En línea]. Disponible en: <https://www.python.org/>
- [5] «PDFBox». [En línea]. Disponible en: <https://pdfbox.apache.org/download.html>
- [6] «PikePDF». [En línea]. Disponible en: <https://pikepdf.readthedocs.io/en/latest/>
- [7] «iText». [En línea]. Disponible en: <https://itextpdf.com/>
- [8] «PyPDF2». [En línea]. Disponible en: <https://pypi.org/project/PyPDF2/>
- [9] D. Senad, «PDF file format: Internal Document Structure Explained», nov. 2022, [En línea]. Disponible en: <https://www.save-emails-as-pdf.com/news/pdf-file-format-internal-document-structure-explained/>
- [10] G. Sánchez Muñoz, «La estructura de los documentos PDF», sep. 2022, [En línea]. Disponible en: [http://www.gusgsm.com/la\\_estructura\\_de\\_los\\_documentos\\_pdf](http://www.gusgsm.com/la_estructura_de_los_documentos_pdf)
- [11] J. Whittington, *PDF explained*, 1st ed. Sebastopol, Calif: O'Reilly Media, 2011.
- [12] «Arquitectura Básica PDF». [En línea]. Disponible en: [https://biblus.us.es/bibing/proyectos/abreproy/11591/fichero/MEMORIA\\_POR\\_VOLUMENES%252F4.+CAPITULO\\_4-+ANTECEDENTES\\_II-ARQUITECTURA\\_B%C3%81SICA\\_DE\\_PDF.pdf+](https://biblus.us.es/bibing/proyectos/abreproy/11591/fichero/MEMORIA_POR_VOLUMENES%252F4.+CAPITULO_4-+ANTECEDENTES_II-ARQUITECTURA_B%C3%81SICA_DE_PDF.pdf+)
- [13] «re library». [En línea]. Disponible en: <https://docs.python.org/3/library/re.html>

## Anexo - Código de la herramienta desarrollado

```

1 import re
2
3 # EXPRESIONES REGULARES
4
5 decimalNumber = r'[\d.]+'
6 naturalNumber = r'-?' + decimalNumber
7
8 font = r'\S*'
9
10 lineFont = r'/( '+font+r') ('+decimalNumber+r') Tf\n'
11 linePos = r'(\d|\s)*?('+naturalNumber+r') ('+naturalNumber+r')
12 (Tm|Td|TD)\n'
13 lineEvery = r'T*\n'
14
15 regexID = r'(\d+) \d+ obj'
16 regexReferencia = r'\d+ \d+ R'
17 regexKids = r'/Kids \[(.*?)\]|/K \[(.*?)\]'
18 regexParent = r'/Parent (\d+) (.*?)'
19 regexContent = r'/Contents (\d+ 0 R)'
20 regexStream = r'stream(.*?)endstream'
21
22 regexObject = r'\d+ \d+ obj(?:\n<<[\s\S]*?endobj|\nendobj)'
23
24 regexContentReference =
25 r'((?!<|[\] (.*?)\n|(<<\n.*?>>)\n|((\[.*?\])\n(/|>>)))'
26 regexReference = r'/(\\S+) '+regexContentReference
27 regexRealContent = r'<<(.*?)>>\nendobj'
28
29 regexTexts = r'(/Span|/P|/LBody|/Artifact) (.*?)EMC'
30
31 regexTextBegin1 = r'/MCID (\d+) (.*?) (Tj|TJ)'
32 regexTextBegin2 = r'/BBox(.*?) (Tj|TJ)'
33
34 # Expresiones regulares para las líneas de impresión del texto
35 regexTextContent1 =
36 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\(((.*?)\n)Tj'
37 regexTextContent2 =
38 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[((?<!\n)\n((.*?)\n)?<!\n)\n)?(-?(\d|\s)*(?!\\)\n(.*?(?!\\)\n))*\n]TJ'
39 regexTextContent3 =
40 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[?[?(<(.*?)>(-
41 (\d|\s)*<.*?>))*\n]?T(J|j)'
42 regexTextContent4 =
43 r'('+lineFont+r')?('+linePos+r')?('+lineEvery+r')?\[-?(\d)\n(\w)\n]TJ'
44
45
46
47
48 # CLASES
49
50 # Clase de objeto del cuerpo del PDF
51 class Object:
52     def __init__(self, content, id):
53         self.id = id

```

```

54         self.content = content
55         self.kids = []
56         self.contents = ""
57         self.printedTexts = []
58         self.references = {}
59
60
61 # Clase de la instrucción de imprimir un texto de un PDF
62 class PrintedText:
63     def __init__(self, id, font, size, rowText, text, posX, posY):
64         self.id = id
65         self.font = font
66         self.size = size
67         self.rowText = rowText
68         self.text = text
69         self.posX = posX
70         self.posY = posY
71
72
73 # MÉTODOS
74
75 # Método que obtiene el id de un objeto
76 def getID(content):
77     line = re.findall(regexID, content)
78     return int(line[0])
79
80
81 # Método que obtiene los hijos de un objeto
82 def getKids(content):
83     match = re.search(regexKids, content)
84     listIDs = []
85
86     if match:
87         reference = match.group()
88         listReferences = re.findall(r'\b(\d+) 0 R\b', reference)
89         listIDs = [int(num) for num in listReferences]
90     return listIDs
91
92
93 # Método que obtiene la referencia de clave /Contents
94 def getContents(content):
95     match = re.search(regexContent, content)
96     if match:
97         reference = match.group(1)
98         contentsStr = reference.split()[0]
99         contentsInt = int(contentsStr)
100     return contentsInt
101
102
103 # Método que obtiene las referencias que están en el diccionario de
104 los objetos
105 def getReferences(content):
106     ref_dic = {}
107     references = re.findall(regexReference, content, re.DOTALL)
108     for ref in references:
109         if ref[2] != '':

```

```

110         ref_dic[ref[0]] = ref[2]
111     elif ref[3] != '':
112         ref_dic[ref[0]] = ref[3]
113     elif ref[5] != '':
114         ref_dic[ref[0]] = ref[5]
115         #print(ref[5])
116     return ref_dic
117
118 # Método que decodifica un texto en hexadecimal a texto ascii
119 def hex_to_ascii(hex_string):
120     bytes_object = bytes.fromhex(hex_string)
121     ascii_string = bytes_object.decode('latin1')
122
123     ascii_string = ascii_string.replace('\x0c', 'f')
124     ascii_string = ascii_string.replace('\r', 'f')
125     return ascii_string
126
127 # Método que comprueba si hay algún caracter especial y lo cambie en
128 una
129 # cadena de texto
130 def search_special_character(text):
131     if text == "f":
132         text = "{"
133     if text == "g":
134         text = "}"
135     if text == "n":
136         text = "\\"
137     if text == "\\003":
138         text = "*"
139     return text
140
141 # Método que comprueba si hay algún caracter especial y lo cambie en
142 una lista
143 # de cadenas de texto
144 def search_spectral_characters(parts):
145     if len(parts) == 1:
146         for i in range(len(parts)):
147             if parts[i] == "f":
148                 parts[i] = "{"
149             if parts[i] == "g":
150                 parts[i] = "}"
151             if parts[i] == "n":
152                 parts[i] = "\\"
153     return parts
154
155 # Método que obtiene los textos que se imprimen en un objeto de tipo
156 # contenido
157 def parse_text(obj, string):
158     txt = PrintedText
159     match1 = re.search(regexTextBegin1, string, re.DOTALL) #
160 Comprueba si coincide
161
162 # con el formato
163 de tecto MCID
164     if match1:
165         id = match1[1] # Obtiene el id de la instrucción
166         aux_text = string[10:]

```

```

166
167         # Encuentra todos los textos que se imprimen dependiendo
168     de
169         # como estén organizadas las cadenas de texto
170         list_aux =
171     re.findall(r'('+regexTextContent1+r')|('+regexTextContent2+r')|('+reg
172     exTextContent3+r')',aux_text, re.DOTALL)
173
174         # Por cada texto y dependiendo del formato recoge el tipo
175         # de fuente, el tamaño, el desplazamiento horizontal y
176     vertical
177         # y el texto que se imprime
178     for aux in list_aux:
179         if aux[0] != '':
180             font = aux[2]
181             size = (aux[3])
182             posX = aux[6]
183             posY = aux[7]
184             rowText = aux[10]
185             text = search_special_character(aux[10])
186             txt = PrintedText(id, font, size, rowText, text,
187     posX, posY)
188         else:
189             if aux[11] != '':
190                 font = aux[13]
191                 size = (aux[14])
192                 posX = aux[17]
193                 posY = aux[18]
194                 rowText = aux[21]
195                 parts = re.findall(r'(?!\)\)\((.*)?(?!\\)\)',
196     aux[21])
197                 parts = search_spectral_characters(parts)
198                 text = ''.join(parts)
199                 txt = PrintedText(id, font, size, rowText, text,
200     posX, posY)
201             else:
202                 if aux[26] != '':
203                     font = aux[28]
204                     size = (aux[29])
205                     posX = aux[32]
206                     posY = aux[33]
207                     rowText = aux[36]
208                     parts = re.findall(r'<(.*?)>', aux[36])
209                     text_parts = [hex_to_ascii(hex) for hex in
210     parts]
211                     text = ''.join(text_parts)
212                     txt = PrintedText(id, font, size, rowText,
213     text, posX, posY)
214
215                 obj.printedTexts.append(txt) # Añade el texto a imprimir
216     en la lista
217
218                                     # de textos del objeto
219         else:
220             match2 = re.search(regexTextBegin2, string, re.DOTALL)
221             if match2:

```

```

222         id = '0'
223         aux_text = string[15:]
224         # Encuentra todos los textos que se imprimen
225         dependiendo de como estén organizadas las cadenas de texto
226         list_aux =
227         re.findall(r'('+regexTextContent1+r')|('+regexTextContent2+r')|('+reg
228         exTextContent3+r')',aux_text, re.DOTALL)
229
230         # Por cada texto y dependiendo del formato recoge el tipo
231         de fuente,
232         # el tamaño, el desplazamiento horizontal y vertical
233         # y el texto que se imprime
234         for aux in list_aux:
235             if aux[0] != '':
236                 font = aux[2]
237                 size = (aux[3])
238                 posX = aux[6]
239                 posY = aux[7]
240                 rowText = aux[10]
241                 text = search_special_character(aux[10])
242                 txt = PrintedText(id, font, size, rowText, text,
243         posX, posY)
244             else:
245                 if aux[11] != '':
246                     font = aux[13]
247                     size = (aux[14])
248                     posX = aux[17]
249                     posY = aux[18]
250                     rowText = aux[21]
251                     parts =
252         re.findall(r'(?<!\)\)\((.*)?(?<!\)\)\)', aux[21])
253                     parts = search_spectral_characters(parts)
254                     text = ''.join(parts)
255                     txt = PrintedText(id, font, size, rowText,
256         text, posX, posY)
257                 else:
258                     if aux[26] != '':
259                         font = aux[28]
260                         size = (aux[29])
261                         posX = aux[32]
262                         posY = aux[33]
263                         rowText = aux[36]
264                         parts = re.findall(r'<(.*?)>', aux[36])
265                         text_parts = [hex_to_ascii(hex) for hex
266         in parts]
267                         text = ''.join(text_parts)
268                         txt = PrintedText(id, font, size,
269         rowText, text, posX, posY)
270
271                 obj.printedTexts.append(txt)         # Añade el texto a
272         imprimir
273
274                 # en la lista de
275         textos del objeto
276
277         # Método que imprime el árbol de estructura desde los objetos de tipo
278         # página hasta los objetos de contenido de las páginas

```



```

278 def print_struct_tree(nodo, prefijo='', es_ultimo=True):
279     print(prefijo, end='')
280     print('└' if es_ultimo else '├', end='')
281     print(nodo.id, end='')
282     print(f' ->{nodo.contents}' if (nodo.contents != None) else '')
283
284     prefijo += '    ' if es_ultimo else '│ '
285     cantidad_hijos = len(nodo.kids)
286
287     for i, hijo in enumerate(nodo.kids):
288         es_ultimo = i == cantidad_hijos - 1
289         print_struct_tree(hijo, prefijo, es_ultimo)
290
291 # Método que imprime el documento PDF entero imprimiendo los textos
292 de
293 # todas las páginas
294 def print_pdf(listContents):
295     for obj in listContents:
296         previous_height = obj.printedTexts[0].posY
297         for text in obj.printedTexts:
298             if (text.posY != ''):
299                 if (abs(float(text.posY) -
300 float(previous_height)) > 10 and text.posY != '0'):
301                     print('\n'+text.text, end='')
302                 else:
303                     print(text.text, end='')
304                     previous_height = text.posY
305             else:
306                 print(text.text, end='')
307
308         print()
309         print()
310
311 # Método que imprime los textos de una de las páginas
312 def print_sheet(obj):
313     previous_height = obj.printedTexts[0].posY
314     for text in obj.printedTexts:
315         if (text.posY != ''):
316             if (abs(float(text.posY) - float(previous_height)) > 10
317 and text.posY != '0'):
318                 print('\n'+text.text, end='')
319             else:
320                 print(text.text, end='')
321                 previous_height = text.posY
322         else:
323             print(text.text, end='')
324
325 # Método que enseña los textos que tiene una página e imprime uno que
326 el
327 # usuario selecciona
328 def show_printed_texts(obj):
329     print()
330     print("Elige un texto con id de los siguientes que quieras
331 imprimir: ")
332     print()
333

```

```

334     n = 0
335     for text in obj.printedTexts:
336         if n == 10:
337             print(f'[{text.id}]')
338             n = 0
339         else:
340             print(f'[{text.id}]', end=', ')
341         n += 1
342
343     # Método que imprime los textos de un objeto cuyo id es uno pasado
344     # como argumento
345     def print_text_by_id(id, obj):
346         print()
347         print(f"El id es {id}")
348         for text in obj.printedTexts:
349             if text.id == id:
350                 print(f'Tipo de fuente:\'{text.font}\',
351 Tamaño:\'{text.size}\', Pos horizontal:\'{text.posX}\', Pos
352 Vertical:\'{text.posY}\', Texto:\'{text.text}\'')
353
354     # Método que imprime las referencias del diccionario de un objeto
355     def print_referencias(obj):
356         print()
357         for clave, valor in obj.references.items():
358             print(f"{clave, valor}")
359         print()
360
361     # Método que modifica el valor de una referencia de un objeto
362     def modificar_referencia(obj):
363         while True:
364             print_referencias(obj)
365
366             clave = input("Escriba la clave del valor que quiere cambiar:
367 ")
368             if clave in obj.references:
369                 antiguo_valor = obj.references[clave]
370                 print(f"\n[{clave} -> {obj.references[clave]}}\n")
371                 nuevo_valor = input("Escriba el nuevo valor de esta
372 referencia. Ten en cuenta de que puedes estropear el PDF (0 para
373 volver): ")
374                 if nuevo_valor != '0':
375
376                     obj.references[clave] = nuevo_valor
377
378                     listCambiosReferencias.append([obj, clave,
379 antiguo_valor, nuevo_valor])
380                     print("Valor cambiado...")
381                     break
382                 else:
383                     break
384             else:
385                 print(f"La clave '{clave}' no existe en el diccionario.")
386
387     # Método que modifica el valor de un texto de un objeto
388     def modificar_texto(id_text, obj):
389         for text in obj.printedTexts:

```

```

390         if text.id == id_text:
391             nuevo_valor = input("Inserte el nuevo valor de este
392 texto: ")
393             antiguo_valor = text.rowText
394             text.text = nuevo_valor
395
396             listCambiosTexts.append([obj, id_text, antiguo_valor,
397 nuevo_valor])
398             print("Valor cambiado...")
399
400 # Método que modifica el contenido de código del documento PDF pasado
401 def modificar_pdf():
402
403     new_content = contenido
404     for cambio in listCambiosReferences:
405         new_content = modificar_pdf_referencias(new_content, cambio)
406
407     for cambio in listCambiosTexts:
408         new_content = modificar_pdf_textos(new_content, cambio)
409
410     return new_content
411
412 # Método que modifica las referencias de un objeto en el contenido de
413 # código del documento PDF pasado
414 def modificar_pdf_referencias(content, cambio):
415     regexObj = str(cambio[0].id)+r" 0 obj.*?endobj"
416
417     match = re.search(regexObj, content, re.DOTALL)
418     if match:
419         last_match = match.group()
420         regexReference = r'/' + str(cambio[1]) + r'
421 '+regexContentReference
422         match1 = re.search(regexReference, match.group())
423         if match1:
424             last_match1 = match1.group()
425             new_match1 = match1.group().replace(cambio[2], cambio[3])
426
427             new_match = match.group().replace(last_match1, new_match1)
428
429             new_content = content.replace(last_match, new_match)
430
431     return new_content
432
433 # Método que modifica los textos de un objeto en el contenido de
434 código
435 # del documento PDF pasado
436 def modificar_pdf_textos(content, cambio):
437     regexObj = str(cambio[0].id)+r" 0 obj.*?endobj"
438     new_content = content
439     match = re.search(regexObj, content, re.DOTALL)
440     if match:
441         last_match = match.group()
442         regexReference = r'MCID ' + str(cambio[1]) + r'.*?EMC'
443         match1 = re.search(regexReference, match.group(), re.DOTALL)
444         if match1:
445             last_match1 = match1.group()

```

```

446         textSearch = cambio[2].replace("(", "\(")
447         textSearch2 = textSearch.replace(")", "\)")
448         regexRowText = r'(\('+textSearch2+r'\))(T(j|J))'
449
450         match2 = re.search(regexRowText, match1.group())
451         if match2:
452             last_match2 = match2.group()
453             newText = "("+cambio[3]+")Tj"
454             new_match1 =
455 match1.group().replace(last_match2, newText, 1)
456
457             new_match =
458 match.group().replace(last_match1, new_match1)
459
460             new_content = content.replace(last_match, new_match)
461         else:
462             regexRowText2 = r'(\['+textSearch2+r'\])(T(j|J))'
463             match3 = re.search(regexRowText2, match1.group())
464             if match3:
465                 last_match3 = match3.group()
466                 newText = "["+cambio[3]+"]TJ"
467                 new_match1 =
468 match1.group().replace(last_match3, newText, 1)
469
470                 new_match =
471 match.group().replace(last_match1, new_match1)
472
473                 new_content =
474 content.replace(last_match, new_match)
475             return new_content
476
477 # Método que imprime los hijos de un objeto
478 def print_kids(obj):
479     print()
480     if obj.kids == []:
481         print("Este objeto no tiene objetos hijos")
482     else:
483         print("El objeto seleccionado tiene los siguientes objetos
484 hijos:")
485         for kid in obj.kids:
486             print(kid.id, end=", ")
487         print()
488         print()
489     return obj.kids
490
491 # Método que elimina una referencia a los hijos de un objeto
492 def delete_kid(obj, kid_id):
493     for kid in obj.kids:
494         if kid.id == kid_id:
495             obj.kids.remove(kid)
496             print(f"Objeto con id {kid.id} eliminado...")
497
498 # Método que añade una referencia a los hijos de un objeto
499 def add_kid(obj, kid_id):
500     obj.kids.append(listObjects[id-1])
501     print(f"Objeto con id {kid_id} añadido...")

```

```

502
503 # Método que gestiona si se quiere añadir, eliminar o ver los hijos
504 # de un objeto
505 def gestionar_hijos(id):
506     kids = print_kids(listObjects[id-1])
507     if kids == []:
508         return
509     else:
510         show_kids_menu()
511         opcion = input("Elija una opción: ")
512         if opcion == '0':
513             return
514         elif opcion == '1':
515             new_id = input("Elija un hijo: ")
516             if any(kid.id == int(new_id) for kid in kids):
517                 delete_kid(listObjects[id-1], int(new_id))
518             else:
519                 print("El id no es no hijo de este objeto")
520                 return
521         elif opcion == '2':
522             new_id = input("Elija un hijo: ")
523             if any(int(new_id) < 2933 and int(new_id) > 0):
524                 add_kid(listObjects[id-1], int(new_id))
525             else:
526                 print("El id no es válido")
527                 return
528         elif opcion == '3':
529             new_id = input("Elija un hijo: ")
530             if any(kid.id == int(new_id) for kid in kids):
531                 gestionar_hijos(int(new_id))
532             else:
533                 print("El id no es no hijo de este objeto")
534                 return
535
536 # Método que muestra el menú de las opciones que se pueden realizar
537 con los hijos de un objeto
538 def show_kids_menu():
539     print("1. Eliminar un hijo")
540     print("2. Añadir un hijo")
541     print("3. Acceder al hijo")
542     print("0. Salir")
543
544 # Abrir el archivo PDF
545 file_path =
546 "c:/Users/34629/Documents/UNIVERSIDAD/TFG/decompressedPDF.txt"
547 try:
548     with open(file_path, 'r', encoding='utf-8', errors='ignore') as
549     archivo:
550         contenido = archivo.read()
551 except UnicodeDecodeError as e:
552     print("Error de decodificación Unicode:", e)
553
554 # Cargar objetos
555 listObjects = []
556 listContents = []
557 listIntContents = []

```

```

558
559 listCambiosReferences = []
560 listCambiosTexts = []
561
562
563 objetos = re.findall(regexObject, contenido)
564
565
566 for objeto in objetos:
567     id = getID(objeto)
568     newObject = Object(objeto,id)
569     listObjects.append(newObject)
570
571 # Cargar objetos
572 for obj in listObjects:
573     children = getKids(obj.content) # Cargar ids de los hijos
574
575     obj.contents = getContents(obj.content)
576     if obj.contents != None:
577         listIntContents.append(obj.contents)
578
579
580     # Cargar referencias
581     obj.references = getReferences(obj.content)
582
583     # Cargar la lista de objetos hijos de un objeto
584     for obj2 in listObjects:
585         for child in children:
586             if obj2.id == child:
587                 obj.kids.append(obj2)
588
589     # Cargar la lista de ids de los objetos de tipo contenido
590     for n in listIntContents:
591         if obj.id == n:
592             listContents.append(obj)
593
594
595     # Cargar los textos de los objetos de tipo contenido
596     for objContent in listContents:
597         lista = re.findall(regexTexts,objContent.content,re.DOTALL)
598         for text in lista:
599             #obj2.printedTexts.append(parse_text(text[1]))
600             parse_text(objContent,text[1])
601
602
603
604 !-- MENU --
605 def exit_menu():
606     print("Saliendo del programa...")
607     exit()
608
609 def option_1():
610     print_struct_tree(listObjects[4])
611
612 def option_2():
613     while True:

```

```

614         pagina = input("Por favor, escribe la página que quieres
615 imprimir (o '0' para volver al menú principal): ")
616         if pagina == '0':
617             print("Volviendo al menú principal...")
618             return
619         try:
620             numero = int(pagina)
621             if 1 <= numero <= 20:
622                 print_sheet(listContents[numero-1])
623                 break
624             else:
625                 print("Número fuera de rango. Por favor, ingresa un
626 número entre 1 y 20.")
627         except ValueError:
628             print("Entrada no válida. Por favor, ingresa un número (o
629 '0' para volver al menú principal): ")
630
631 def option_3():
632     print_pdf(listContents)
633
634 def option_4():
635     while True:
636         pagina = input("Por favor, escribe la página del texto que
637 quieres imprimir (o '0' para volver al menú principal): ")
638         if pagina == '0':
639             print("Volviendo al menú principal...")
640             return
641         try:
642             numero = int(pagina)
643             if 1 <= numero <= 20:
644                 show_printed_texts(listContents[numero-1])
645                 id_text = input("\nSelecciona un id para imprimir: ")
646                 try:
647                     print_text_by_id(id_text, listContents[numero-1])
648                     break
649                 except:
650                     print("Entrada no válida, no has insertado un id
651 válido para etsa página")
652             else:
653                 print("Número fuera de rango. Por favor, ingresa un
654 número entre 1 y 20.")
655         except ValueError:
656             print("Entrada no válida. Por favor, ingresa un número (o
657 '0' para volver al menú principal): ")
658
659 def option_5():
660     while True:
661         id = input("Por favor, escribe el id del objeto. Debe ser un
662 número del 1 al 2933 (o '0' para volver al menú principal): ")
663         if id == '0':
664             print("Volviendo al menú principal...")
665             return
666         try:
667             numero = int(id)
668             if 1 <= numero <= 2933:
669                 print_referencias(listObjects[numero-1])

```

```

670         break
671     else:
672         print("Número fuera de rango. Por favor, ingresa un
673 número entre 1 y 2933.")
674     except ValueError:
675         print("Entrada no válida. Por favor, ingresa un número
676 entre 1 y 2933 (o '0' para volver al menú principal): ")
677
678 def option_6():
679     while True:
680         print()
681         print("0. Volver la menú principal")
682         print("1. Modificar una referencia de un objeto")
683         print("2. Modificar un texto de un objeto")
684
685         opcion = input("Selecciona una opción (0-2): ")
686
687         if opcion == '0':
688             print("Volviendo al menú principal...")
689             return
690
691
692         elif opcion == '1':
693             id_objeto = input("Por favor, escribe el id del objeto.
694 Debe ser un número del 1 al 2933 (o '0' para volver al menú
695 principal): ")
696             if id_objeto == '0':
697                 print("Volviendo al menú principal...")
698                 return
699             try:
700                 numero = int(id_objeto)
701                 if 1 <= numero <= 2933:
702                     modificar_referencia(listObjects[numero-1])
703                     break
704                 else:
705                     print("Número fuera de rango. Por favor, ingresa
706 un número entre 1 y 2933.")
707             except ValueError:
708                 print("Entrada no válida. Por favor, ingresa un
709 número entre 1 y 2933 (o '0' para volver al menú principal): ")
710
711
712         elif opcion == '2':
713
714             pagina = input("Por favor, escribe la página del texto
715 que quieres imprimir (o '0' para volver al menú principal): ")
716             if pagina == '0':
717                 print("Volviendo al menú principal...")
718                 return
719             try:
720                 numero = int(pagina)
721                 if 1 <= numero <= 20:
722                     show_printed_texts(listContents[numero-1])
723                     id_text = input("\nSelecciona un id para
724 modificar: ")
725                     try:

```



```

726             print_text_by_id(id_text,
727 listContents[numero-1])
728
729             modificar_texto(id_text, listContents[numero-
730 1])
731
732             break
733         except:
734             print("Entrada no válida, no has insertado un
735 id válido para etsa página")
736         else:
737             print("Número fuera de rango. Por favor, ingresa
738 un número entre 1 y 20.")
739         except ValueError:
740             print("Entrada no válida. Por favor, ingresa un
741 número (o '0' para volver al menú principal): ")
742
743         else:
744             print("Entrada no válida. Inserte un número del 0 al 2")
745
746
747 def option_7():
748
749     new_content = modificar_pdf()
750
751
752     new_file_path =
753 "c:/Users/34629/Documents/UNIVERSIDAD/TFG/newFile.txt"
754     with open(new_file_path, 'w', encoding='utf-8') as archivo:
755         archivo.write(new_content)
756     print()
757     print("Nuevo archivo creado...")
758     print()
759
760 def option_8():
761     while True:
762         id = input("Por favor, escribe el id del objeto. Debe ser un
763 número del 1 al 2933 (o '0' para volver al menú principal): ")
764         if id == '0':
765             print("Volviendo al menú principal...")
766             return
767         try:
768             numero = int(id)
769             if 1 <= numero <= 2933:
770                 gestionar_hijos(numero)
771                 break
772             else:
773                 print("Número fuera de rango. Por favor, ingresa un
774 número entre 1 y 2933.")
775             except ValueError:
776                 print("Entrada no válida. Por favor, ingresa un número
777 entre 1 y 2933 (o '0' para volver al menú principal): ")
778
779 # Método que muestra el menú de opciones
780 def show_menu():
781     print("-----")

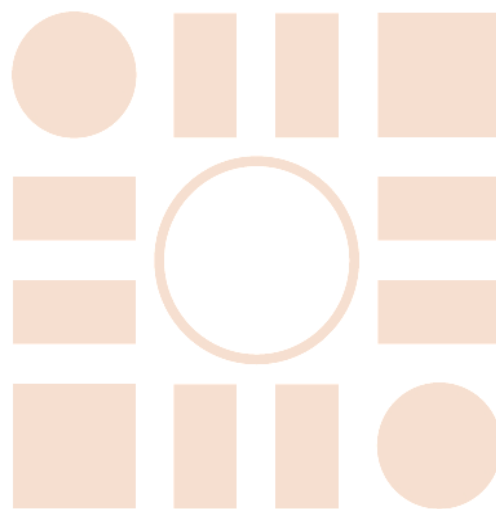
```

```

782  -----")
783      print("\nMenú de opciones")
784      print("1. Imprimir el árbol de estructura del PDF")
785      print("2. Imprimir una página del PDF")
786      print("3. Imprimir el PDF entero")
787      print("4. Imprimir un texto por id")
788      print("5. Imprimir referencias de un objeto")
789      print("6. Modificar objeto")
790      print("7. Descargar PDF")
791      print("8. Gestionar los hijos de un objeto")
792      print("9. Salir")
793      print("-----")
794  -----")
795
796  # Método principal del programa
797  def main():
798      while True:
799          show_menu()
800          opcion = input("Selecciona una opción (1-9): ")
801
802          if opcion == '1':
803              option_1()
804          elif opcion == '2':
805              option_2()
806          elif opcion == '3':
807              option_3()
808          elif opcion == '4':
809              option_4()
810          elif opcion == '5':
811              option_5()
812          elif opcion == '6':
813              option_6()
814          elif opcion == '7':
815              option_7()
816          elif opcion == '8':
817              option_8()
818          elif opcion == '9':
819              exit_menu()
820          else:
821              print("Opción no válida. Por favor, intenta de nuevo.")
822          print()
823
824  # Llamada al método principal del programa
825  if __name__ == "__main__":
826      main()

```

Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR



Universidad  
de Alcalá