pypdf
Release 3.17.0

Mathieu Fenniak

Oct 29, 2023

USER GUIDE

pypdf is a free and open source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. pypdf can retrieve text and metadata from PDFs as well.

See pdfly for a CLI application that uses pypdf to interact with PDFs.

You can contribute to pypdf on GitHub.

INSTALLATION

There are several ways to install pypdf. The most common option is to use pip.

1.1 pip

pypdf requires Python 3.6+ to run.

Typically Python comes with `pip`, a package installer. Using it you can install pypdf:

```
pip install pypdf
```

If you are not a super-user (a system administrator / root), you can also just install pypdf for your current user:

```
pip install --user pypdf
```

1.1.1 Optional dependencies

pypdf tries to be as self-contained as possible, but for some tasks the amount of work to properly maintain the code would be too high. This is especially the case for cryptography and image formats.

If you simply want to install all optional dependencies, run:

```
pip install pypdf[full]
```

Alternatively, you can install just some:

If you plan to use pypdf for encrypting or decrypting PDFs that use AES, you will need to install some extra dependencies. Encryption using RC4 is supported using the regular installation.

```
pip install pypdf[crypto]
```

If you plan to use image extraction, you need Pillow:

```
pip install pypdf[image]
```

1.2 Python Version Support

Python	3.11	3.10	3.9	3.8	3.7	3.6	2.7
pypdf>=3.0	YES	YES	YES	YES	YES	YES	
PyPDF2>=2.0	YES	YES	YES	YES	YES	YES	
PyPDF2 1.20.0 - 1.28.4		YES	YES	YES	YES	YES	YES
PyPDF2 1.15.0 - 1.20.0							YES

1.3 Anaconda

Anaconda users can install pypdf via conda-forge.

1.4 Development Version

In case you want to use the current version under development:

```
pip install git+https://github.com/py-pdf/pypdf.git
```

MIGRATION GUIDE: 1.X TO 2.X

PyPDF2<`2.0.0` (docs) is very different from PyPDF2>=`2.0.0` (*docs*).

Luckily, most changes are simple naming adjustments. This guide helps you to make the step from PyPDF2 `1.x` (or even the original PyPpdf) to PyPDF2>=`2.0.0`.

You can execute your code with the updated version and show deprecation warnings by running `python -W all your_code.py`.

THREE

IMPORTS AND MODULES

- `PyPDF2.utils` no longer exists

- `PyPDF2.pdf` no longer exists. You can import from `PyPDF2` directly or from `PyPDF2.generic`

FOUR

NAMING ADJUSTMENTS

4.1 Classes

The base classes were renamed as they also allow to operate with ByteIO streams instead of files. Also, the `strict` parameter changed the default value from `strict=True` to `strict=False`.

- `PdfFileReader PdfReader`
- `PdfFileWriter PdfWriter`
- `PdfFileMerger PdfMerger`

PdfFileReader and PdfFileMerger no longer have the `overwriteWarnings` parameter. The new behavior is `overwriteWarnings=False`.

4.2 Function, Method, and Property Names

In `PyPDF2.xmp.XmpInformation`:

- `rdfRoot rdf_root`
- `xmp_createDate xmp_create_date`
- `xmp_creatorTool xmp_creator_tool`
- `xmp_metadataDate xmp_metadata_date`
- `xmp_modifyDate xmp_modify_date`
- `xmpMetadata xmp_metadata`
- `xmpmm_documentId xmpmm_document_id`
- `xmpmm_instanceId xmpmm_instance_id`

In `PyPDF2.generic`:

- `readObject read_object`
- `convertToInt convert_to_int`
- `DocumentInformation.getText DocumentInformation._get_text` : This method should typically not be used; please let me know if you need it.
- `readHexStringFromStream read_hex_string_from_stream`
- `initializeFromDictionary initialize_from_dictionary`
- `createStringObject create_string_object`

- `TreeObject.hasChildren` `TreeObject.has_children`

- `TreeObject.emptyTree` `TreeObject.empty_tree`

In many places:

- `getObject` `get_object`

- `writeToStream` `write_to_stream`

- `readFromStream` `read_from_stream`

PdfReader class:

- `reader.getPage(pageNumber)` `reader.pages[page_number]`

- `reader.getNumPages()` / `reader.numPages` `len(reader.pages)`

- `getDocumentInfo` `metadata`

- `flattenedPages` attribute `flattened_pages`

- `resolvedObjects` attribute `resolved_objects`

- `xrefIndex` attribute `xref_index`

- `getNamedDestinations` / `namedDestinations` attribute `named_destinations`

- `getPageLayout` / `pageLayout` `page_layout` attribute

- `getPageMode` / `pageMode` `page_mode` attribute

- `getIsEncrypted` / `isEncrypted` `is_encrypted` attribute

- `getOutlines` `get_outlines`

- `readObjectHeader` `read_object_header`

- `cacheGetIndirectObject` `cache_get_indirect_object`

- `cacheIndirectObject` `cache_indirect_object`

- `getDestinationPageNumber` `get_destination_page_number`

- `readNextEndLine` `read_next_end_line`

- `_zeroXref` `_zero_xref`

- `_authenticateUserPassword` `_authenticate_user_password`

- `_pageId2Num` attribute `_page_id2num`

- `_buildDestination` `_build_destination`

- `_buildOutline` `_build_outline`

- `_getPageNumberByIndirect(indirectRef)` `_get_page_number_by_indirect(indirect_ref)`

- `_getObjectFromStream` `_get_object_from_stream`

- `_decryptObject` `_decrypt_object`

- `_flatten(..., indirectRef)` `_flatten(..., indirect_ref)`

- `_buildField` `_build_field`

- `_checkKids` `_check_kids`

- `_writeField` `_write_field`

- `_write_field(..., fieldAttributes)` `_write_field(..., field_attributes)`

- `_read_xref_subsections(..., getEntry, ...)` `_read_xref_subsections(..., get_entry, ...)`

PdfWriter class:

- `writer.getPage(pageNumber)` `writer.pages[page_number]`
- `writer.getNumPages()` `len(writer.pages)`
- `addMetadata` `add_metadata`
- `addPage` `add_page`
- `addBlankPage` `add_blank_page`
- `addAttachment(fname, fdata)` `add_attachment(filename, data)`
- `insertPage` `insert_page`
- `insertBlankPage` `insert_blank_page`
- `appendPagesFromReader` `append_pages_from_reader`
- `updatePageFormFieldValues` `update_page_form_field_values`
- `cloneReaderDocumentRoot` `clone_reader_document_root`
- `cloneDocumentFromReader` `clone_document_from_reader`
- `getReference` `get_reference`
- `getOutlineRoot` `get_outline_root`
- `getNamedDestRoot` `get_named_dest_root`
- `addBookmarkDestination` `add_bookmark_destination`
- `addBookmarkDict` `add_bookmark_dict`
- `addBookmark` `add_bookmark`
- `addNamedDestinationObject` `add_named_destination_object`
- `addNamedDestination` `add_named_destination`
- `removeLinks` `remove_links`
- `removeImages(ignoreByteStringObject)` `remove_images(ignore_byte_string_object)`
- `removeText(ignoreByteStringObject)` `remove_text(ignore_byte_string_object)`
- `addURI` `add_uri`
- `addLink` `add_link`
- `getPage(pageNumber)` `get_page(page_number)`
- `getPageLayout / setPageLayout / pageLayout` `page_layout attribute`
- `getPageMode / setPageMode / pageMode` `page_mode attribute`
- `_addObject` `_add_object`
- `_addPage` `_add_page`
- `_sweepIndirectReferences` `_sweep_indirect_references`

PdfMerger class

- `__init__` parameter: `strict=True` `strict=False` (the `PdfFileMerger` still has the old default)

- `addMetadata` `add_metadata`

- `addNamedDestination` `add_named_destination`

- `setPageLayout` `set_page_layout`

- `setPageMode` `set_page_mode`

Page class:

- `artBox` / `bleedBox` / `cropBox` / `mediaBox` / `trimBox` `artbox` / `bleedbox` / `cropbox` / `mediabox` / `trimbox`

  - `getWidth`, `getHeight` `width` / `height`

  - `getLowerLeft_x` / `getUpperLeft_x` `left`

  - `getUpperRight_x` / `getLowerRight_x` `right`

  - `getLowerLeft_y` / `getLowerRight_y` `bottom`

  - `getUpperRight_y` / `getUpperLeft_y` `top`

  - `getLowerLeft` / `setLowerLeft` `lower_left` property

  - `upperRight` `upper_right`

- `mergePage` `merge_page`

- `rotateClockwise` / `rotateCounterClockwise` `rotate_clockwise`

- `_mergeResources` `_merge_resources`

- `_contentStreamRename` `_content_stream_rename`

- `_pushPopGS` `_push_pop_gs`

- `_addTransformationMatrix` `_add_transformation_matrix`

- `_mergePage` `_merge_page`

XmpInformation class:

- `getElement(..., aboutUri, ...)` `get_element(..., about_uri, ...)`

- `getNodesInNamespace(..., aboutUri, ...)` `get_nodes_in_namespace(..., aboutUri, ...)`

- `_getText` `_get_text`

utils.py:

- `matrixMultiply` `` `matrix_multiply ``

- `RC4_encrypt` is moved to the security module

4.3 Parameter Names

- `PdfWriter.get_page`: `pageNumber` `page_number`

- `PyPDF2.filters` (all classes): `decodeParms` `decode_parms`

- `PyPDF2.filters` (all classes): `decodeStreamData` `decode_stream_data`

- `pagenum` `page_number`

- `PdfMerger.merge`: `position` `page_number`

- `PdfWriter.add_outline_item_destination`: `dest` `page_destination`

- `PdfWriter.add_named_destination_object`: dest page_destination
- `PdfWriter.encrypt`: user_pwd user_password
- `PdfWriter.encrypt`: owner_pwd owner_password

4.4 Deprecations

A few classes / functions were deprecated without replacement:

- `PyPDF2.utils.ConvertFunctionsToVirtualList`
- `PyPDF2.utils.formatWarning`
- `PyPDF2.isInt(obj)`: Use `instance(obj, int)` instead
- `PyPDF2.u_(s)`: Use s directly
- `PyPDF2.chr_(c)`: Use `chr(c)` instead
- `PyPDF2.barray(b)`: Use `bytearray(b)` instead
- `PyPDF2.isBytes(b)`: Use `instance(b, type(bytes()))` instead
- `PyPDF2.xrange_fn`: Use `range` instead
- `PyPDF2.string_type`: Use `str` instead
- `PyPDF2.isString(s)`: Use `instance(s, str)` instead
- `PyPDF2._basestring`: Use `str` instead
- `b_(...)` was removed. You should typically be able use the bytes object directly, otherwise you can copy this

ROBUSTNESS AND STRICT=FALSE

PDF is specified in various versions. The specification of PDF 1.7 has 978 pages. This length makes it hard to get everything right. As a consequence, a lot of PDF files are not strictly following the specification.

If a PDF file does not follow the specification, it is not always possible to be certain what the intended effect would be. Think of the following broken Python code as an example:

```python
# Broken
function (foo, bar):

# Potentially intended:
def function(foo, bar):
    ...

# Also possible:
function = (foo, bar)
```

Writing a parser you can go two paths: Either you try to be forgiving and try to figure out what the user intended, or you are strict and just tell the user that they should fix their stuff.

pypdf gives you the option to be strict or not.

pypdf has three core objects and all of them have a `strict` parameter:

- *PdfReader*

- *PdfWriter*

- *PdfMerger*

Choosing `strict=True` means that pypdf will raise an exception if a PDF does not follow the specification.

Choosing `strict=False` means that pypdf will try to be forgiving and do something reasonable, but it will log a warning message. It is a best-effort approach.

EXCEPTIONS, WARNINGS, AND LOG MESSAGES

pypdf makes use of 3 mechanisms to show that something went wrong:

- **Log messages** are informative messages that can be used for post-mortem analysis. Most of the time, users can ignore them. They come in different *levels*, such as info / warning / error indicating the severity. Examples are non-standard compliant PDF files which pypdf can deal with.

- **Warnings** are avoidable issues, such as using deprecated classes / functions / parameters. Another example is missing capabilities of pypdf. In those cases, pypdf users should adjust their code. Warnings are issued by the `warnings` module - those are different from the log-level "warning".

- **Exceptions** are error-cases that pypdf users should explicitly handle. In the `strict=True` mode, most log messages with the warning level will become exceptions. This can be useful in applications where you can force to user to fix the broken PDF.

6.1 Exceptions

Exeptions need to be catched if you want to handle them. For example, you could want to read the text from a PDF as a part of a search function.

Most PDF files don't follow the specifications. In this case pypdf needs to guess which kinds of mistakes were potentially done when the PDF file was created. See *the robustness page* for the related issues.

As a users, you likely don't care about it. If it's readable in any way, you want the text. You might use pdfminer.six as a fallback and do this:

```python
from pypdf import PdfReader
from pdfminer.high_level import extract_text as fallback_text_extraction

text = ""
try:
    reader = PdfReader("example.pdf")
    for page in reader.pages:
        text += page.extract_text()
except Exception as exc:
    text = fallback_text_extraction("example.pdf")
```

You could also capture `pypdf.errors.PyPdfError` if you prefer something more specific.

6.2 Warnings

The `warnings` module allows you to ignore warnings:

```python
import warnings

warnings.filterwarnings("ignore")
```

In many cases, you actually want to start Python with the `-W` flag so that you see all warnings. This is especially true for Continuous Integration (CI).

6.3 Log messages

Log messages can be noisy in some cases. pypdf hopefully is having a reasonable level of log messages, but you can reduce which types of messages you want to see:

```python
import logging

logger = logging.getLogger("pypdf")
logger.setLevel(logging.ERROR)
```

The `logging` module defines six log levels:

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

METADATA

7.1 Reading metadata

```python
from pypdf import PdfReader

reader = PdfReader("example.pdf")

meta = reader.metadata

print(len(reader.pages))

# All of the following could be None!
print(meta.author)
print(meta.creator)
print(meta.producer)
print(meta.subject)
print(meta.title)
```

7.2 Writing metadata

```python
from datetime import datetime
from pypdf import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# If you want to add the old metadata, include this line
metadata = reader.metadata
writer.add_metadata(metadata)

# Format the current date and time for the metadata
utc_time = "-05'00'"  # UTC time optional
time = datetime.now().strftime(f"D\072%Y%m%d%H%M%S{utc_time}")
```

```python
# Add the new metadata
writer.add_metadata(
    {
        "/Author": "Martin",
        "/Producer": "Libre Writer",
        "/Title": "Title",
        "/Subject": "Subject",
        "/Keywords": "Keywords",
        "/CreationDate": time,
        "/ModDate": time,
        "/Creator": "Creator",
        "/CustomField": "CustomField",
    }
)

# Save the new PDF to a file
with open("meta-pdf.pdf", "wb") as f:
    writer.write(f)
```

EXTRACT TEXT FROM A PDF

You can extract text from a PDF like this:

```python
from pypdf import PdfReader

reader = PdfReader("example.pdf")
page = reader.pages[0]
print(page.extract_text())
```

you can also choose to limit the text orientation you want to extract, e.g:

```python
# extract only text oriented up
print(page.extract_text(0))

# extract text oriented up and turned left
print(page.extract_text((0, 90)))
```

Refer to extract_text for more details.

8.1 Using a visitor

You can use visitor-functions to control which part of a page you want to process and extract. The visitor-functions you provide will get called for each operator or for each text fragment.

The function provided in argument visitor_text of function extract_text has five arguments:

- text: the current text (as long as possible, can be up to a full line)
- user_matrix: current matrix to move from user coordinate space (also known as CTM)
- tm_matrix: current matrix from text coordinate space
- font-dictionary: full font dictionary
- font-size: the size (in text coordinate space)

The matrix stores 6 parameters. The first 4 provide the rotation/scaling matrix and the last two provide the translation (horizontal/vertical) It is recommended to use the user_matrix as it takes into all transformations.

Notes :

- as indicated in the PDF 1.7 reference, page 204 the user matrix applies to text space/image space/form space/pattern space.

- if you want to get the full transformation from text to user space, you can use the `mult` function (availalbe in global import) as follows: `txt2user = mult(tm, cm))` The font-size is the raw text size, that is affected by the `user_matrix`

The font-dictionary may be None in case of unknown fonts. If not None it may e.g. contain key "/BaseFont" with value "/Arial,Bold".

Caveat: In complicated documents the calculated positions may be difficult to (if you move from multiple forms to page user space for example).

The function provided in argument visitor_operand_before has four arguments: operator, operand-arguments, current transformation matrix and text matrix.

8.1.1 Example 1: Ignore header and footer

The following example reads the text of page 4 of this PDF document, but ignores header (y < 720) and footer (y > 50).

```python
from pypdf import PdfReader

reader = PdfReader("GeoBase_NHNC1_Data_Model_UML_EN.pdf")
page = reader.pages[3]

parts = []


def visitor_body(text, cm, tm, font_dict, font_size):
    y = cm[5]
    if y > 50 and y < 720:
        parts.append(text)


page.extract_text(visitor_text=visitor_body)
text_body = "".join(parts)

print(text_body)
```

8.1.2 Example 2: Extract rectangles and texts into a SVG-file

The following example converts page 3 of this PDF document into a SVG file.

Such a SVG export may help to understand whats going on in a page.

```python
from pypdf import PdfReader
import svgwrite

reader = PdfReader("GeoBase_NHNC1_Data_Model_UML_EN.pdf")
page = reader.pages[2]

dwg = svgwrite.Drawing("GeoBase_test.svg", profile="tiny")


def visitor_svg_rect(op, args, cm, tm):
    if op == b"re":
```

```python
        (x, y, w, h) = (args[i].as_numeric() for i in range(4))
        dwg.add(dwg.rect((x, y), (w, h), stroke="red", fill_opacity=0.05))


def visitor_svg_text(text, cm, tm, fontDict, fontSize):
    (x, y) = (cm[4], cm[5])
    dwg.add(dwg.text(text, insert=(x, y), fill="blue"))


page.extract_text(
    visitor_operand_before=visitor_svg_rect, visitor_text=visitor_svg_text
)
dwg.save()
```

The SVG generated here is bottom-up because the coordinate systems of PDF and SVG differ.

Unfortunately in complicated PDF documents the coordinates given to the visitor-functions may be wrong.

8.2 Why Text Extraction is hard

8.2.1 Unclear Objective

Extracting text from a PDF can be pretty tricky. In several cases there is no clear answer what the expected result should look like:

1. **Paragraphs**: Should the text of a paragraph have line breaks at the same places where the original PDF had them or should it rather be one block of text?

2. **Page numbers**: Should they be included in the extract?

3. **Headers and Footers**: Similar to page numbers - should they be extracted?

4. **Outlines**: Should outlines be extracted at all?

5. **Formatting**: If text is **bold** or *italic*, should it be included in the output?

6. **Tables**: Should the text extraction skip tables? Should it extract just the text? Should the borders be shown in some Markdown-like way or should the structure be present e.g. as an HTML table? How would you deal with merged cells?

7. **Captions**: Should image and table captions be included?

8. **Ligatures**: The Unicode symbol U+FB00 is a single symbol ﬀ for two lowercase letters 'f'. Should that be parsed as the Unicode symbol 'ﬀ' or as two ASCII symbols 'ff'?

9. **SVG images**: Should the text parts be extracted?

10. **Mathematical Formulas**: Should they be extracted? Formulas have indices, and nested fractions.

11. **Whitespace characters**: How many new lines should be extracted for 3cm of vertical whitespace? How many spaces should be extracted if there is 3cm of horizontal whitespace? When would you extract tabs and when spaces?

12. **Footnotes**: When the text of multiple pages is extracted, where should footnotes be shown?

13. **Hyperlinks and Metadata**: Should it be extracted at all? Where should it be placed in which format?

14. **Linearization**: Assume you have a floating figure in between a paragraph. Do you first finish the paragraph or do you put the figure text in between?

Then there are issues where most people would agree on the correct output, but the way PDF stores information just makes it hard to achieve that:

1. **Tables**: Typically, tables are just absolutely positioned text. In the worst case, ever single letter could be absolutely positioned. That makes it hard to tell where columns / rows are.

2. **Images**: Sometimes PDFs do not contain the text as it's displayed, but instead an image. You notice that when you cannot copy the text. Then there are PDF files that contain an image and a text layer in the background. That typically happens when a document was scanned. Although the scanning software (OCR) is pretty good today, it still fails once in a while. pypdf is no OCR software; it will not be able to detect those failures. pypdf will also never be able to extract text from images.

And finally there are issues that pypdf will deal with. If you find such a text extraction bug, please share the PDF with us so we can work on it!

8.2.2 Missing Semantic Layer

The PDF file format is all about producing the desired visual result for printing. It was not created for parsing the content. PDF files don't contain a semantic layer.

Specifically, there is no information what the header, footer, page numbers, tables, and paragraphs are. The visual appearence is there and people might find heuristics to make educated guesses, but there is no way of being certain.

This is a shortcoming of the PDF file format, not of pypdf.

It would be possible to apply machine learning on PDF documents to make good heuristics, but that will not be part of pypdf. However, pypdf could be used to feed such a machine learning system with the relevant information.

8.2.3 Whitespaces

The PDF format is meant for printing. It is not designed to be read by machines. The text within a PDF document is absolutely positioned, meaning that every single character could be positioned on the page.

The text

> This is a test document by Ethan Nelson.

can be represented as

> [(This is a)9(te)-3(st)9(do)-4(cu)13(m)-4(en)12(t)-3(b)3(y)-3( )9(Et)-2(h)3(an)4(Nels)13(o)-5(n)3(.)]
> TJ

Where the numbers are adjustments of vertical space. This representation used within the PDF file makes it very hard to guarantee correct whitespaces.

More information:

- issue #1507
- Negative numbers in PDF content stream text object
- Mark Stephens: Understanding PDF text objects, 2010.

8.3 OCR vs Text Extraction

Optical Character Recognition (OCR) is the process of extracting text from images. Software which does this is called *OCR software*. The tesseract OCR engine is the most commonly known Open Source OCR software.

pypdf is **not** OCR software.

8.3.1 Digitally-born vs Scanned PDF files

PDF documents can contain images and text. PDF files don't store text in a semantically meaningful way, but in a way that makes it easy to show the text on screen or print it. For this reason text extraction from PDFs is hard.

If you scan a document, the resulting PDF typically shows the image of the scan. Scanners then also run OCR software and put the recognized text in the background of the image. This result of the scanners OCR software can be extracted by pypdf. However, in such cases it's recommended to directly use OCR software as errors can accumulate: The OCR software is not perfect in recognizing the text. Then it stores the text in a format that is not meant for text extraction and pypdf might make mistakes parsing that.

Hence I would distinguish three types of PDF documents:

- **Digitally-born PDF files**: The file was created digitally on the computer. It can contain images, texts, links, outline items (a.k.a., bookmarks), JavaScript, … If you Zoom in a lot, the text still looks sharp.

- **Scanned PDF files**: Any number of pages was scanned. The images were then stored in a PDF file. Hence the file is just a container for those images. You cannot copy the text, you don't have links, outline items, JavaScript.

- **OCRed PDF files**: The scanner ran OCR software and put the recognized text in the background of the image. Hence you can copy the text, but it still looks like a scan. If you zoom in enough, you can recognize pixels.

8.3.2 Can we just always use OCR?

You might now wonder if it makes sense to just always use OCR software. If the PDF file is digitally-born, you can just render it to an image.

I would recommend not to do that.

Text extraction software like pypdf can use more information from the PDF than just the image. It can know about fonts, encodings, typical character distances and similar topics.

That means pypdf has a clear advantage when it comes to characters which are easy to confuse such as oO0öö. **pypdf will never confuse characters**. It just reads what is in the file.

pypdf also has an edge when it comes to characters which are rare, e.g. . OCR software will not be able to recognize smileys correctly.

8.4 Attempts to prevent text extraction

If people who share PDF documents want to prevent text extraction, there are multiple ways to do so:

1. Store the contents of the PDF as an image

2. Use a scrambled font

However, text extraction cannot be completely prevented if people should still be able to read the document. In the worst case people can make a screenshot, print it, scan it, and run OCR over it.

POST-PROCESSING IN TEXT EXTRACTION

Post-processing can recognizably improve the results of text extraction. It is, however, outside of the scope of pypdf itself. Hence the library will not give any direct support for it. It is a natural language processing (NLP) task.

This page lists a few examples what can be done as well as a community recipie that can be used as a best-practice general purpose post processing step. If you know more about the specific domain of your documents, e.g. the language, it is likely that you can find custom solutions that work better in your context

9.1 Ligature Replacement

```python
def replace_ligatures(text: str) -> str:
    ligatures = {
        "ff": "ff",
        "fi": "fi",
        "fl": "fl",
        "ffi": "ffi",
        "ffl": "ffl",
        "ſt": "ft",
        "st": "st",
        # "": "AA",
        # "Æ": "AE",
        "": "aa",
    }
    for search, replace in ligatures.items():
        text = text.replace(search, replace)
    return text
```

9.2 De-Hyphenation

Hyphens are used to break words up so that the appearance of the page is nicer.

```python
from typing import List


def remove_hyphens(text: str) -> str:
    """

    This fails for:
```

(continues on next page)

```python
    * Natural dashes: well-known, self-replication, use-cases, non-semantic,
                      Post-processing, Window-wise, viewpoint-dependent
    * Trailing math operands: 2 - 4
    * Names: Lopez-Ferreras, VGG-19, CIFAR-100
    """
    lines = [line.rstrip() for line in text.split("\n")]

    # Find dashes
    line_numbers = []
    for line_no, line in enumerate(lines[:-1]):
        if line.endswith("-"):
            line_numbers.append(line_no)

    # Replace
    for line_no in line_numbers:
        lines = dehyphenate(lines, line_no)

    return "\n".join(lines)


def dehyphenate(lines: List[str], line_no: int) -> List[str]:
    next_line = lines[line_no + 1]
    word_suffix = next_line.split(" ")[0]

    lines[line_no] = lines[line_no][:-1] + word_suffix
    lines[line_no + 1] = lines[line_no + 1][len(word_suffix) :]
    return lines
```

9.3 Header/Footer Removal

The following header/footer removal has several drawbacks:

- False-positives, e.g. for the first page when there is a date like 2021.

- False-negatives in many cases:

    - Dynamic part, e.g. page label is in the header

    - Even/odd pages have different headers

    - Some pages, e.g. the first one or chapter pages, don't have a header

```python
def remove_footer(extracted_texts: list[str], page_labels: list[str]):
    def remove_page_labels(extracted_texts, page_labels):
        processed = []
        for text, label in zip(extracted_texts, page_labels):
            text_left = text.lstrip()
            if text_left.startswith(label):
                text = text_left[len(label) :]

            text_right = text.rstrip()
            if text_right.endswith(label):
```

```
            text = text_right[: -len(label)]

        processed.append(text)
    return processed

extracted_texts = remove_page_labels(extracted_texts, page_labels)
return extracted_texts
```

9.4 Other ideas

- Whitespaces between Units: Between a number and it's unit should be a space (source). That means: 42 ms, 42 GHz, 42 GB.

- Percent: English style guides prescribe writing the percent sign following the number without any space between (e.g. 50%).

- Whitespaces before dots: Should typically be removed

- Whitespaces after dots: Should typically be added

TEN

EXTRACT IMAGES

Please note: In order to use the following code you need to install optional dependencies, see *installation guide*.

Every page of a PDF document can contain an arbitrary amount of images. The names of the files may not be unique.

```python
from pypdf import PdfReader

reader = PdfReader("example.pdf")

page = reader.pages[0]
count = 0

for image_file_object in page.images:
    with open(str(count) + image_file_object.name, "wb") as fp:
        fp.write(image_file_object.data)
        count += 1
```

ELEVEN

EXTRACT ATTACHMENTS

PDF documents can contain attachments. Attachments have a name, but it might not be unique. For this reason, the value of `reader.attachments["attachment_name"]` is a list.

You can extract all attachments like this:

```python
from pypdf import PdfReader

reader = PdfReader("example.pdf")

for name, content_list in reader.attachments:
    for i, content in enumerate(content_list):
        with open(f"{name}-{i}", "wb") as fp:
            fp.write(content)
```

ENCRYPTION AND DECRYPTION OF PDFS

PDF encryption makes use of RC4 and AES algorithms with different key length. `pypdf` supports all of them until `PDF-2.0`, which is the latest PDF standard.

`pypdf` use an extra dependency to do encryption or decryption for `AES` algorithms. We recommend pyca/cryptography. For Python 3.6 users, you can use pycryptodome as a fallback.

> Please see the note in the *installation guide* for installing the extra dependencies if interacting with PDFs that use AES.

12.1 Encrypt

You can encrypt a PDF by using a password:

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Add a password to the new PDF
writer.encrypt("my-secret-password", algorithm="AES-256")

# Save the new PDF to a file
with open("encrypted-pdf.pdf", "wb") as f:
    writer.write(f)
```

The algorithm can be one of `RC4-40`, `RC4-128`, `AES-128`, `AES-256-R5`, `AES-256`. We recommend using `AES-256-R5`.

> WARNING : pypdf uses RC4 by default for compatibility if you omit the "algorithm" parameter. Since RC4 is insecure, you should use `AES` algorithms.

12.2 Decrypt

You can decrypt a PDF using the appropriate password:

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("encrypted-pdf.pdf")
writer = PdfWriter()

if reader.is_encrypted:
    reader.decrypt("my-secret-password")

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Save the new PDF to a file
with open("decrypted-pdf.pdf", "wb") as f:
    writer.write(f)
```

MERGING PDF FILES

13.1 Basic Example

```python
from pypdf import PdfWriter

merger = PdfWriter()

for pdf in ["file1.pdf", "file2.pdf", "file3.pdf"]:
    merger.append(pdf)

merger.write("merged-pdf.pdf")
merger.close()
```

For more details, see an excellent answer on StackOverflow by Paul Rooney.

13.2 Showing more merging options

```python
from pypdf import PdfWriter

merger = PdfWriter()

input1 = open("document1.pdf", "rb")
input2 = open("document2.pdf", "rb")
input3 = open("document3.pdf", "rb")

# Add the first 3 pages of input1 document to output
merger.append(fileobj=input1, pages=(0, 3))

# Insert the first page of input2 into the output beginning after the second page
merger.merge(position=2, fileobj=input2, pages=(0, 1))

# Append entire input3 document to the end of the output document
merger.append(input3)

# Write to an output PDF document
output = open("document-output.pdf", "wb")
merger.write(output)
```

```python
# Close file descriptors
merger.close()
output.close()
```

13.3 append

append has been slightly extended in `PdfWriter`. See PdfWriter.append for more details.

13.3.1 Examples

```python
# Append the first 10 pages of source.pdf
writer.append("source.pdf", (0, 10))

# Append the first and 10th page from reader and create an outline
writer.append(reader, "page 1 and 10", [0, 9])
```

During merging, the relevant named destination will also imported.

If you want to insert pages in the middle of the destination, use `merge` (which provides (insertion) position). You can insert the same page multiple times, if necessary even using a list-based syntax:

```python
writer.append(reader, [0, 1, 0, 2, 0])
```

will insert the pages 1 and 2 with page 0 before, in the middle and after.

13.4 add_page / insert_page

It is recommended to use `append` or `merge` instead.

13.5 Merging forms

When merging forms, some form fields may have the same names, preventing access to some data.

A grouping field should be added before adding the source PDF to prevent that. The original fields will be identified by adding the group name.

For example, after calling `reader.add_form_topname("form1")`, the field previously named `field1` will now identified as `form1.field1` when calling `reader.get_form_text_fields(True)` or `reader.get_fields()`.

After that, you can append the input PDF completely or partially using `writer.append` or `writer.merge`. If you insert a set of pages, only those fields will be listed.

13.6 reset_translation

During cloning, if an object has been already cloned, it will not be cloned again, and a pointer to this previously cloned object is returned instead. Because of that, if you add/merge a page that has already been added, the same object will be added the second time. If you modify any of these two pages later, both pages can be modified independently.

To reset, call `writer.reset_translation(reader)`.

13.7 Advanced cloning

In order to prevent side effects between pages/objects and all objects linked cloning is done during the merge.

This process will be automatically applied if you use `PdfWriter.append/merge/add_page/insert_page`. If you want to clone an object before attaching it "manually", use the `clone` method of any *PdfObject*:

```
cloned_object = object.clone(writer)
```

If you try to clone an object already belonging to the writer, it will return the same object:

```
assert cloned_object == object.clone(writer)
```

The same holds true if you try to clone an object twice. It will return the previously cloned object:

```
assert object.clone(writer) == object.clone(writer)
```

Please note that if you clone an object, you will clone all the objects below as well, including the objects pointed by *IndirectObject*. Due to this, if you clone a page that includes some articles ("/B"), not only the first article, but also all the chained articles and the pages where those articles can be read will be copied. It means that you may copy lots of objects which will be saved in the output PDF as well.

In order to prevent this, you can provide the list of fields in the dictionaries to be ignored:

```
new_page = writer.add_page(reader.pages[0], excluded_fields=["/B"])
```

13.7.1 Merging rotated pages

If you are working with rotated pages, you might want to call `transfer_rotation_to_content()` on the page before merging to avoid wrongly rotated results:

```
for page in writer.pages:
    if page.rotation != 0:
        page.transfer_rotation_to_content()
    page.merge_page(background, over=False)
```

CROPPING AND TRANSFORMING PDFS

Notice: Just because content is no longer visible, it is not gone. Cropping works by adjusting the viewbox. That means content that was cropped away can still be restored.

```python
from pypdf import PdfWriter, PdfReader

reader = PdfReader("example.pdf")
writer = PdfWriter()

# add page 1 from reader to output document, unchanged:
writer.add_page(reader.pages[0])

# add page 2 from reader, but rotated clockwise 90 degrees:
writer.add_page(reader.pages[1].rotate(90))

# add page 3 from reader, but crop it to half size:
page3 = reader.pages[2]
page3.mediabox.upper_right = (
    page3.mediabox.right / 2,
    page3.mediabox.top / 2,
)
writer.add_page(page3)

# add some Javascript to launch the print window on opening this PDF.
# the password dialog may prevent the print dialog from being shown,
# comment the the encription lines, if that's the case, to try this out:
writer.add_js("this.print({bUI:true,bSilent:false,bShrinkToFit:true});")

# write to document-output.pdf
with open("pypdf-output.pdf", "wb") as fp:
    writer.write(fp)
```

14.1 Page rotation

The most typical rotation is a clockwise rotation of the page by multiples of 90 degrees. That is done when the orientation of the page is wrong. You can do that with the `rotate` method of the `PageObject` class:

```python
from pypdf import PdfWriter, PdfReader

reader = PdfReader("input.pdf")
writer = PdfWriter()


writer.add_page(reader.pages[0])
writer.pages[0].rotate(90)

with open("output.pdf", "wb") as fp:
    writer.write(fp)
```

The rotate method is typically preferred over the `page.add_transformation(Transformation().rotate())` method, because `rotate` will ensure that the page is still in the mediabox / cropbox. The transformation object operates on the coordinates of the pages contents and does not change the mediabox or cropbox.

14.2 Plain Merge
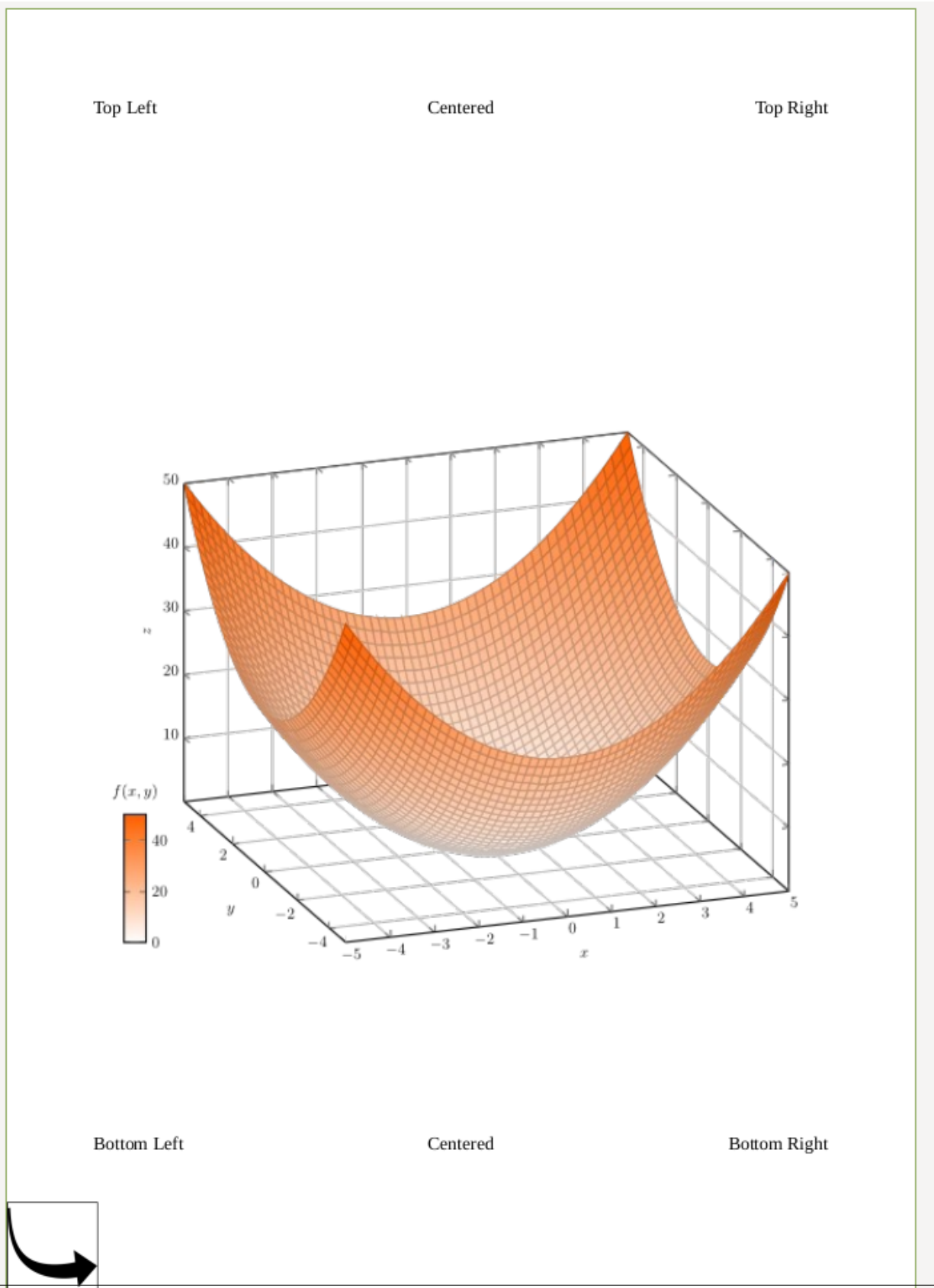
Top Left   Centered   Top Right

Bottom Left   Centered   Bottom Right

is the result of

```python
from pypdf import PdfReader, PdfWriter, Transformation

# Get the data
reader_base = PdfReader("labeled-edges-center-image.pdf")
page_base = reader_base.pages[0]

reader = PdfReader("box.pdf")
page_box = reader.pages[0]

page_base.merge_page(page_box)

# Write the result back
writer = PdfWriter()
writer.add_page(page_base)
with open("merged-foo.pdf", "wb") as fp:
    writer.write(fp)
```

14.3 Merge with Rotation
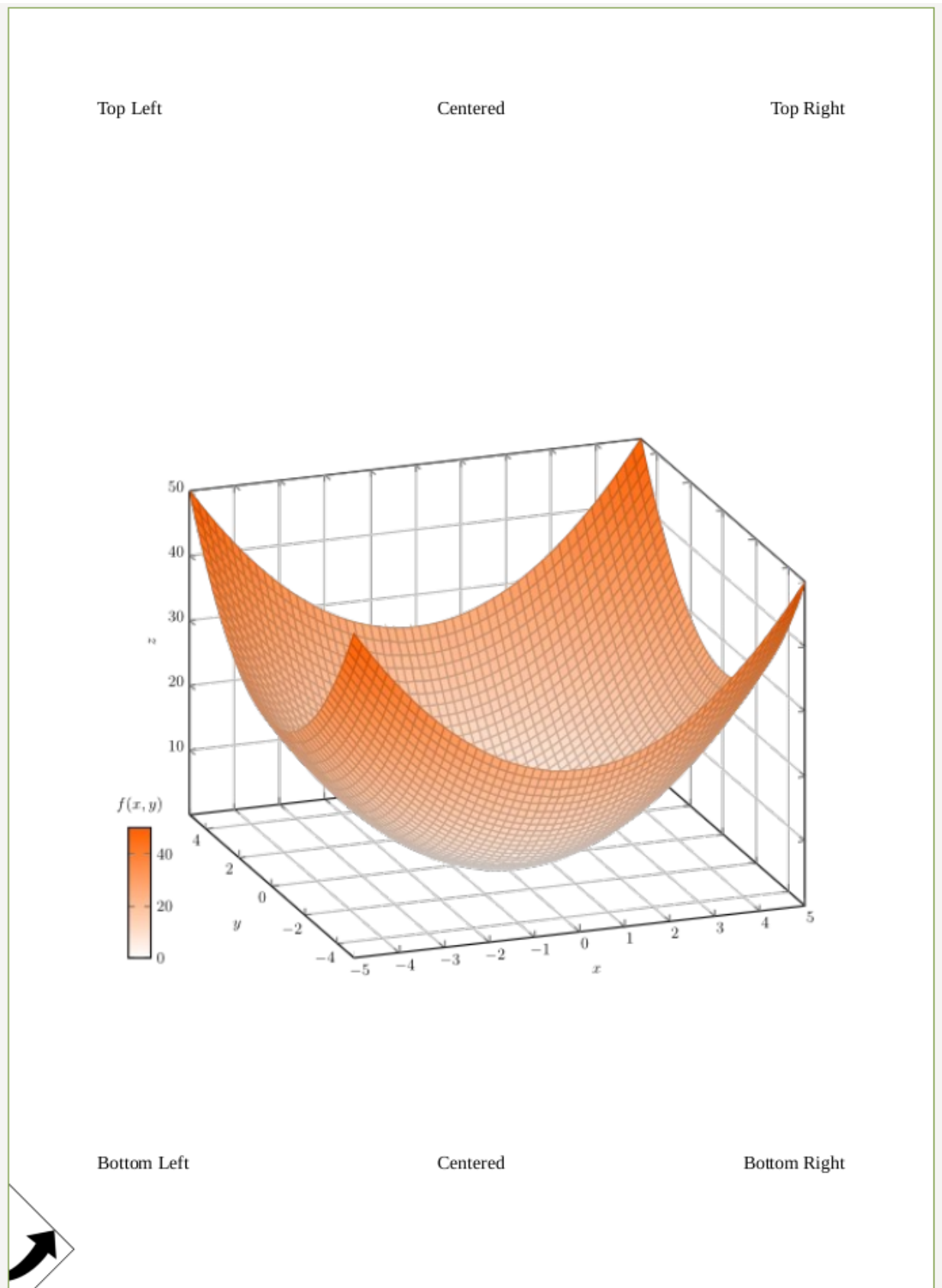
Top Left   Centered   Top Right

Bottom Left   Centered   Bottom Right

```python
from pypdf import PdfReader, PdfWriter, Transformation

# Get the data
reader_base = PdfReader("labeled-edges-center-image.pdf")
page_base = reader_base.pages[0]

reader = PdfReader("box.pdf")
page_box = reader.pages[0]

# Apply the transformation
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)

# Write the result back
writer = PdfWriter()
writer.add_page(page_base)
with open("merged-foo.pdf", "wb") as fp:
    writer.write(fp)
```

If you add the expand parameter:

```python
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)
```

you get:
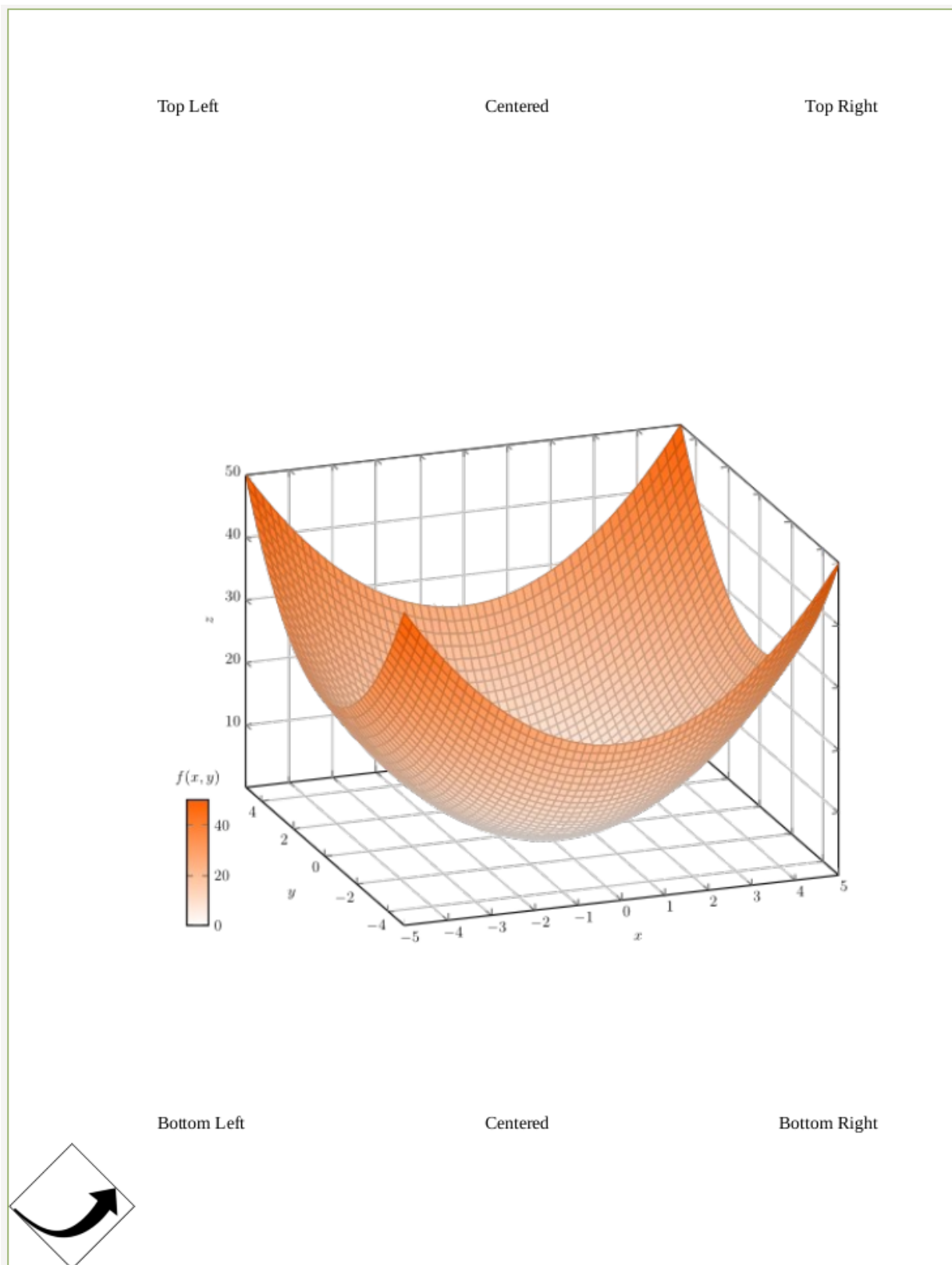
Alternatively, you can move the merged image a bit to the right by using

```
op = Transformation().rotate(45).translate(tx=50)
```

Top Left   Centered   Top Right

Bottom Left   Centered   Bottom Right
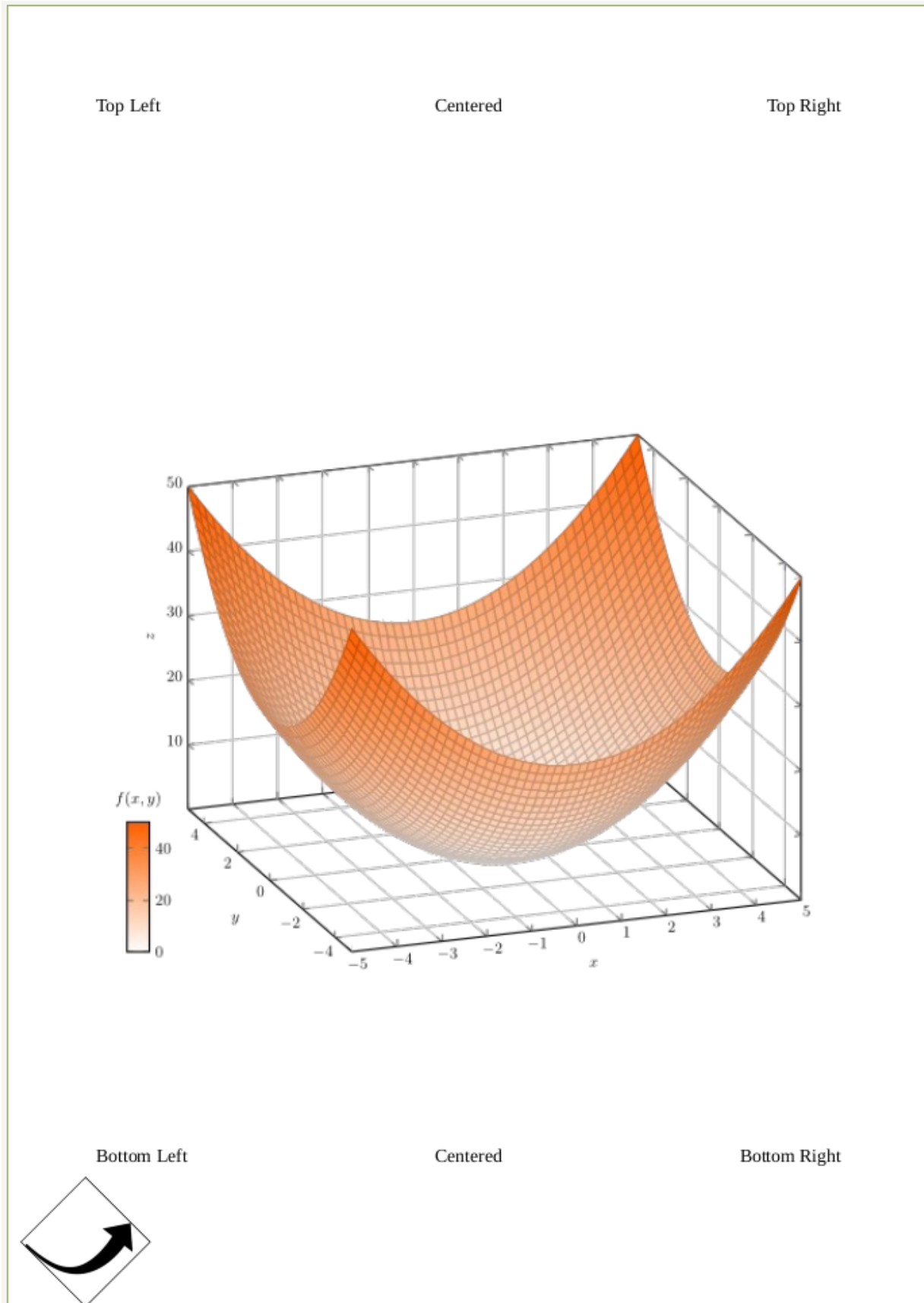
14.4 Scaling

pypdf offers two ways to scale: The page itself and the contents on a page. Typically, you want to combine both.
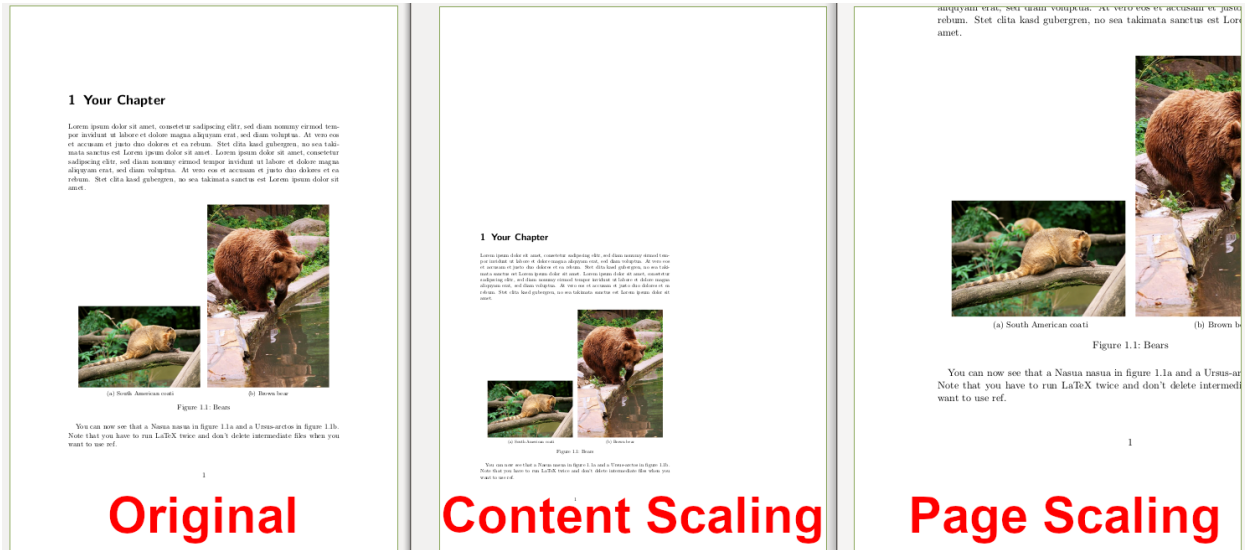
14.4.1 Scaling a Page (the Canvas)

```python
from pypdf import PdfReader, PdfWriter

# Read the input
reader = PdfReader("resources/side-by-side-subfig.pdf")
page = reader.pages[0]

# Scale
page.scale_by(0.5)

# Write the result to a file
writer = PdfWriter()
writer.add_page(page)
writer.write("out.pdf")
```

If you wish to have more control, you can adjust the various page boxes directly:

```python
from pypdf.generic import RectangleObject

mb = page.mediabox

page.mediabox = RectangleObject((mb.left, mb.bottom, mb.right, mb.top))
page.cropbox = RectangleObject((mb.left, mb.bottom, mb.right, mb.top))
page.trimbox = RectangleObject((mb.left, mb.bottom, mb.right, mb.top))
page.bleedbox = RectangleObject((mb.left, mb.bottom, mb.right, mb.top))
page.artbox = RectangleObject((mb.left, mb.bottom, mb.right, mb.top))
```

14.4.2 Scaling the content

The content is scaled towards the origin of the coordinate system. Typically, that is the lower-left corner.

```python
from pypdf import PdfReader, PdfWriter, Transformation

# Read the input
reader = PdfReader("resources/side-by-side-subfig.pdf")
page = reader.pages[0]

# Scale
op = Transformation().scale(sx=0.7, sy=0.7)
page.add_transformation(op)

# Write the result to a file
writer = PdfWriter()
writer.add_page(page)
writer.write("out-pg-transform.pdf")
```

14.4.3 pypdf._page.MERGE_CROP_BOX

pypdf<=3.4.0 used to merge the other page with `trimbox`.

pypdf>3.4.0 changes this behavior to `cropbox`.

In case anybody has good reasons to use/expect `trimbox`, please let me know via info@martin-thoma.de or via https://github.com/py-pdf/pypdf/pull/1622 In the mean time, you can add the following code to get the old behavior:

```python
pypdf._page.MERGE_CROP_BOX = "trimbox"
```

TRANSFORMING SEVERAL COPIES OF THE SAME PAGE

We have designed the following business card (A8 format) to advertize our new startup.

We would like to copy this card sixteen times on an A4 page, to print it, cut it, and give it to all our friends. Having learned about the `merge_page()` method and the `Transformation` class, we run the following code. Notice that we had to tweak the media box of the source page to extend it, which is already a dirty hack (in this case).

```python
from pypdf import PdfReader, PdfWriter, Transformation, PaperSize

# Read source file
reader = PdfReader("nup-source.pdf")
sourcepage = reader.pages[0]

# Create a destination file, and add a blank page to it
writer = PdfWriter()
destpage = writer.add_blank_page(width=PaperSize.A4.height, height=PaperSize.A4.width)

# Extend source page mediabox
sourcepage.mediabox = destpage.mediabox

# Copy source page to destination page, several times
for x in range(4):
    for y in range(4):
        # Translate page
        sourcepage.add_transformation(
            Transformation().translate(
                x * PaperSize.A8.height,
                y * PaperSize.A8.width,
            )
        )
        # Merge translated page
        destpage.merge_page(sourcepage)

# Write file
with open("nup-dest1.pdf", "wb") as fp:
    writer.write(fp)
```

And the result is... unexpected.

The problem is that, having run `add.transformation()` several times on the *same* source page, those transformations add up: for instance, the sixteen transformations are applied to the last copy of the source page, so most of the business cards are *outside* the destination page.

We need a way to merge a transformed page, *without* modifying the source page. Here comes `merge_transformed_page()`. With this method:

- we no longer need the media box hack of our first try;

- transformations are only applied *once*.

```python
from pypdf import PdfReader, PdfWriter, Transformation, PaperSize

# Read source file
reader = PdfReader("nup-source.pdf")
sourcepage = reader.pages[0]

# Create a destination file, and add a blank page to it
writer = PdfWriter()
destpage = writer.add_blank_page(width=PaperSize.A4.height, height=PaperSize.A4.width)

# Copy source page to destination page, several times
for x in range(4):
    for y in range(4):
        destpage.merge_transformed_page(
            sourcepage,
            Transformation().translate(
                x * sourcepage.mediabox.width,
                y * sourcepage.mediabox.height,
            ),
```

```
        )

# Write file
with open("nup-dest2.pdf", "wb") as fp:
    writer.write(fp)
```

We get the expected result.

There is still some work to do, for instance to insert margins between and around cards, but this is left as an exercise for the reader. . .

ADDING A STAMP/WATERMARK TO A PDF

Adding stamps or watermarks are two common ways to manipulate PDF files. A stamp is adding something on top of the document, a watermark is in the background of the document.

16.1 Stamp (Overlay) / Watermark (Underlay)

The process of stamping and watermarking is the same, you just need to set `over` parameter to `True` for stamping and `False` for watermarking.

You can use `merge_page()` if you don't need to transform the stamp:

```python
from pypdf import PdfWriter, PdfReader

stamp = PdfReader("bg.pdf").pages[0]
writer = PdfWriter(clone_from="source.pdf")
for page in writer.pages:
    page.merge_page(stamp, over=False)  # here set to False for watermarking

writer.write("out.pdf")
```

Else use `merge_transformed_page()` with `Transformation()` if you need to translate, rotate, scale, etc. the stamp before merging it to the content page.

```python
from pathlib import Path
from typing import Union, Literal, List

from pypdf import PdfWriter, PdfReader


def stamp(
    content_pdf: Path,
    stamp_pdf: Path,
    pdf_result: Path,
    page_indices: Union[Literal["ALL"], List[int]] = "ALL",
):
    stamp_page = PdfReader(stamp_pdf).pages[0]

    writer = PdfWriter()
    # page_indices can be a List(array) of page, tuples are for range definition
    writer.append(content, pages=None if page_indices == "ALL" else page_indices)
```

(continues on next page)

```python
    for content_page in writer.pages:
        content_page.merge_transformed_page(
            stamp_page,
            Transformation().scale(0.5),
        )

    writer.write(pdf_result)
```

If you are experiencing wrongly rotated watermarks/stamps, try to use `transfer_rotation_to_content()` on the corresponding pages beforehand to fix the page boxes.

Example of stamp:

Chapter 16. Adding a Stamp/Watermark to a PDF

The Crazy Ones

October 14, 1998

Heres to the crazy ones. The misfits. The rebels. The troublemakers.
The round pegs in the square holes.

The ones who see things differently. Theyre not fond of rules. And
they have no respect for the status quo. You can quote them,
disagree with them, glorify or vilify them.

About the only thing you cant do is ignore them. Because they change
things. They invent. They imagine. They heal. They explore. They
create. They inspire. They push the human race forward.

Maybe they have to be crazy.

How else can you stare at an empty canvas and see a work of art? Or
sit in silence and hear a song thats never been written? Or gaze at
a red planet and see a laboratory on wheels?

We make tools for these kinds of people.

While some see them as the crazy ones, we see genius. Because the
people who are crazy enough to think they can change the world,
are the ones who do.

Example of watermark:

SEVENTEEN

READING PDF ANNOTATIONS

PDF 1.7 defines 25 different annotation types:

- Text

- Link

- FreeText

- Line, Square, Circle, Polygon, PolyLine, Highlight, Underline, Squiggly, StrikeOut

- Stamp, Caret, Ink

- Popup

- FileAttachment

- Sound, Movie

- Widget, Screen

- PrinterMark

- TrapNet

- Watermark

- 3D

In general, annotations can be read like this:

```python
from pypdf import PdfReader

reader = PdfReader("commented.pdf")

for page in reader.pages:
    if "/Annots" in page:
        for annot in page["/Annots"]:
            obj = annot.get_object()
            annotation = {"subtype": obj["/Subtype"], "location": obj["/Rect"]}
            print(annotation)
```

Reading the most common ones is described here.

17.1 Text

```
from pypdf import PdfReader

reader = PdfReader("example.pdf")

for page in reader.pages:
    if "/Annots" in page:
        for annot in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/Text":
                print(annot.get_object()["/Contents"])
```

17.2 Highlights

```
from pypdf import PdfReader

reader = PdfReader("commented.pdf")

for page in reader.pages:
    if "/Annots" in page:
        for annot in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/Highlight":
                coords = annot.get_object()["/QuadPoints"]
                x1, y1, x2, y2, x3, y3, x4, y4 = coords
```

17.3 Attachments

```
from pypdf import PdfReader

reader = PdfReader("example.pdf")

attachments = {}
for page in reader.pages:
    if "/Annots" in page:
        for annotation in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/FileAttachment":
                fileobj = annotobj["/FS"]
                attachments[fileobj["/F"]] = fileobj["/EF"]["/F"].get_data()
```

ADDING PDF ANNOTATIONS

18.1 Attachments

```python
from pypdf import PdfWriter

writer = PdfWriter()
writer.add_blank_page(width=200, height=200)

data = b"any bytes - typically read from a file"
writer.add_attachment("smile.png", data)

with open("output.pdf", "wb") as output_stream:
    writer.write(output_stream)
```
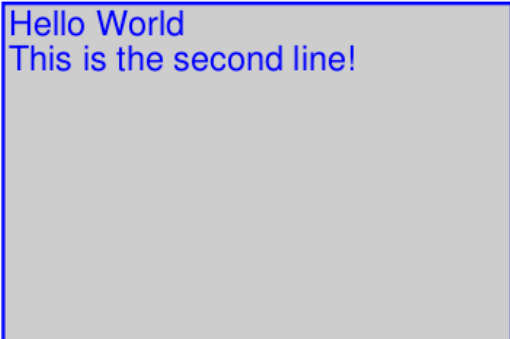
18.2 Free Text

If you want to add text in a box like this

The Crazy Ones

October 14, 1998

Heres to the crazy ones. The misfits. The rebels. The troublemakers.
The round pegs in the square holes.

Hello World
This is the second line!

~~erently. Theyre not fond of rules. And~~
~~he status quo. You can quote them,~~
~~y or vilify them.~~

~~nt do is ignore them. Because they change~~
~~y imagine. They heal. They explore. They~~
~~y push the human race forward.~~

How else can you stare at an empty canvas and see a work of art? Or
sit in silence and hear a song thats never been written? Or gaze at
a red planet and see a laboratory on wheels?

We make tools for these kinds of people.

While some see them as the crazy ones, we see genius. Because the
people who are crazy enough to think they can change the world,
are the ones who do.

you can use the *FreeText*:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import FreeText

# Fill the writer with the pages you want
pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Create the annotation and add it
annotation = FreeText(
    text="Hello World\nThis is the second line!",
    rect=(50, 550, 200, 650),
    font="Arial",
    bold=True,
    italic=True,
    font_size="20pt",
    font_color="00ff00",
```

(continues on next page)

```
    border_color="0000ff",
    background_color="cdcdcd",
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

18.3 Text

A text annotation looks like this:

Contents

1	Foo	2
2	Bar	2
3	Baz	2
4	Foo	2
5	Bar	3
6	Baz	3
7	Foo	3
8	Bar	4
9	Baz	4

18.4 Line

If you want to add a line like this:

The Crazy Ones

October 14, 1998

Heres to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes.

The ones who see things differently. Theyre not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them.

About the only thing you cant do is ignore them. Because they change things. They invent. They imagine. They heal. They explore. They create. They inspire. They push the human race forward.

Maybe they have to be crazy.

How else can you stare at an empty canvas and see a work of art? Or sit in silence and hear a song thats never been written? Or gaze at a red planet and see a laboratory on wheels?

We make tools for these kinds of people.

While some see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

you can use *Line*:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Line

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the line
annotation = Line(
    text="Hello World\nLine2",
    rect=(50, 550, 200, 650),
    p1=(50, 550),
    p2=(200, 650),
)
writer.add_annotation(page_number=0, annotation=annotation)
```
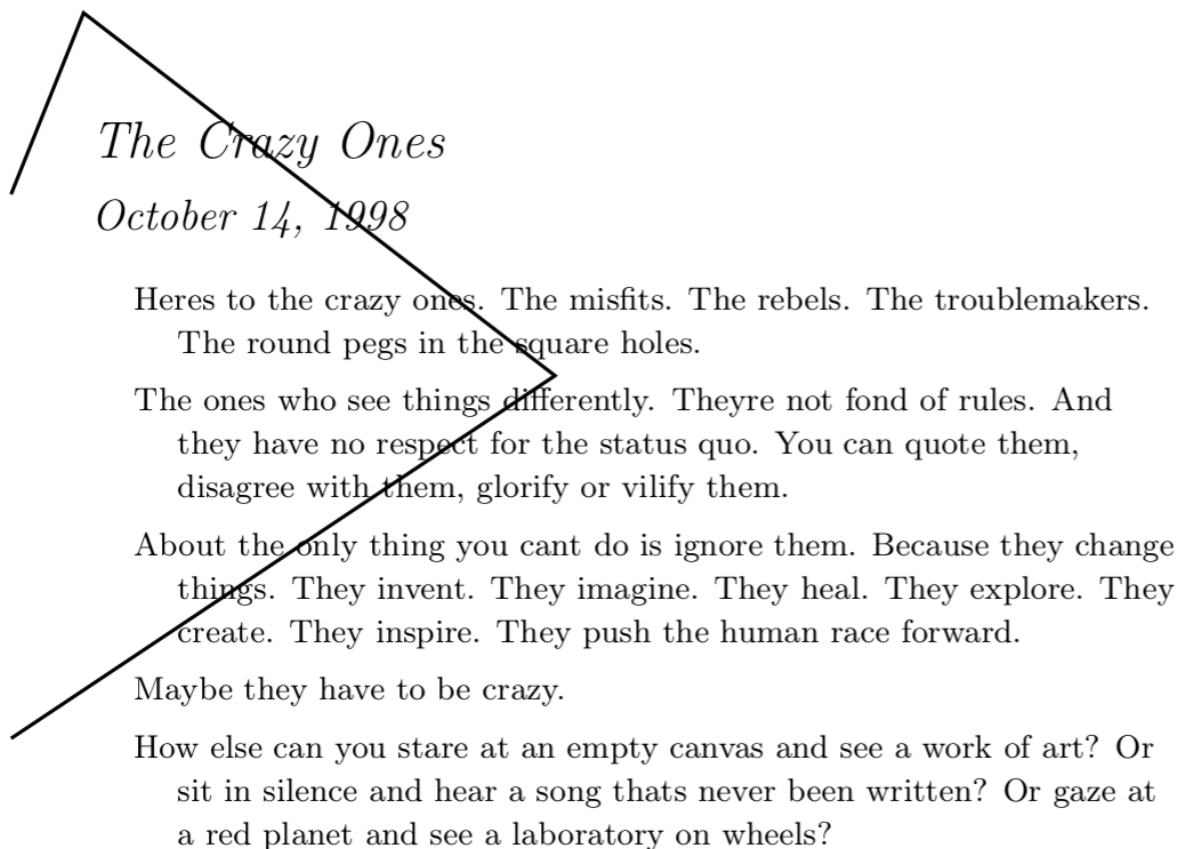
```python
# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

18.5 PolyLine

If you want to add a line like this:

The Crazy Ones

October 14, 1998

Heres to the crazy ones. The misfits. The rebels. The troublemakers.
The round pegs in the square holes.

The ones who see things differently. Theyre not fond of rules. And
they have no respect for the status quo. You can quote them,
disagree with them, glorify or vilify them.

About the only thing you cant do is ignore them. Because they change
things. They invent. They imagine. They heal. They explore. They
create. They inspire. They push the human race forward.

Maybe they have to be crazy.

How else can you stare at an empty canvas and see a work of art? Or
sit in silence and hear a song thats never been written? Or gaze at
a red planet and see a laboratory on wheels?

you can use *PolyLine*:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Polyline

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the polyline
```

```
annotation = Polyline(
    vertices=[(50, 550), (200, 650), (70, 750), (50, 700)],
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

18.6 Rectangle

If you want to add a rectangle like this:

The Crazy Ones

October 14, 1998

Heres to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes.

The ones who see things differently. Theyre not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them.

About the only thing you cant do is ignore them. Because they change things. They invent. They imagine. They heal. They explore. They create. They inspire. They push the human race forward.

Maybe they have to be crazy.

How else can you stare at an empty canvas and see a work of art? Or sit in silence and hear a song thats never been written? Or gaze at a red planet and see a laboratory on wheels?

We make tools for these kinds of people.

you can use *Rectangle*:

```
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Rectangle

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the rectangle
annotation = Rectangle(
    rect=(50, 550, 200, 650),
```
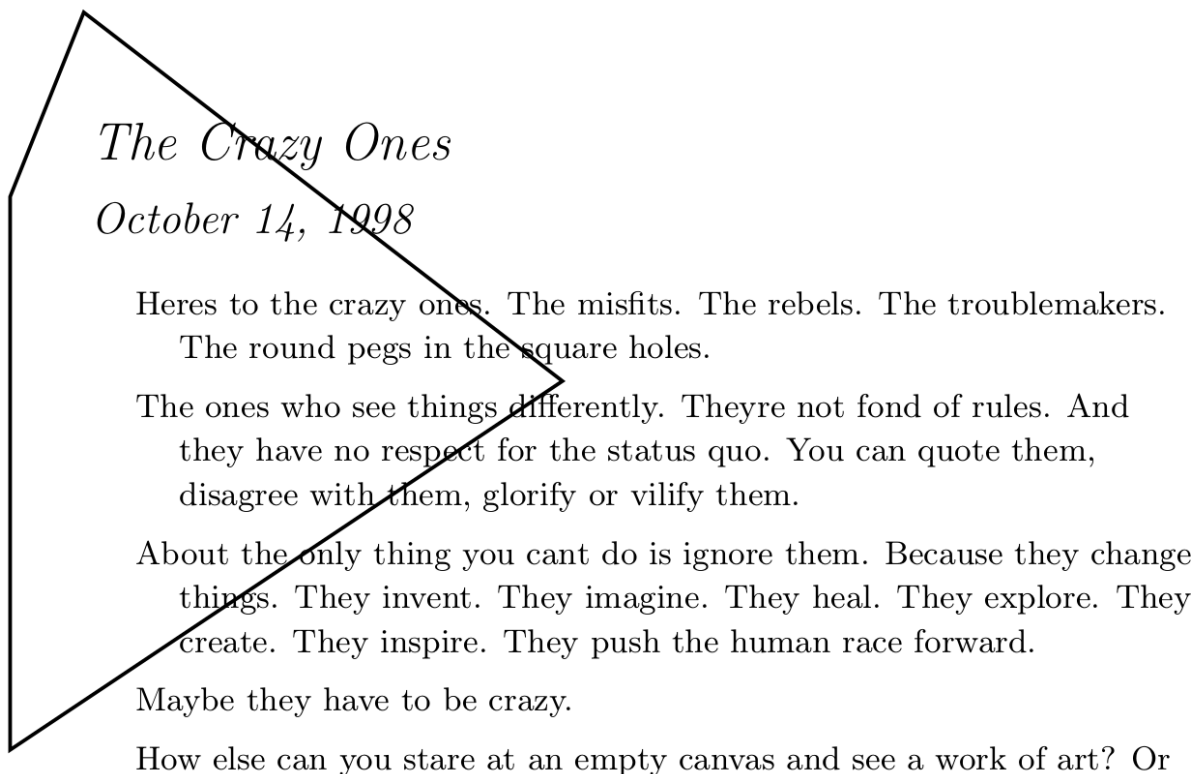
```
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

If you want the rectangle to be filled, use the `interiour_color="ff0000"` parameter.

This method uses the "square" annotation type of the PDF format.

18.7 Ellipse

If you want to add a circle like this:

you can use *Ellipse*:

```
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Ellipse

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the rectangle
annotation = Ellipse(
    rect=(50, 550, 200, 650),
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```
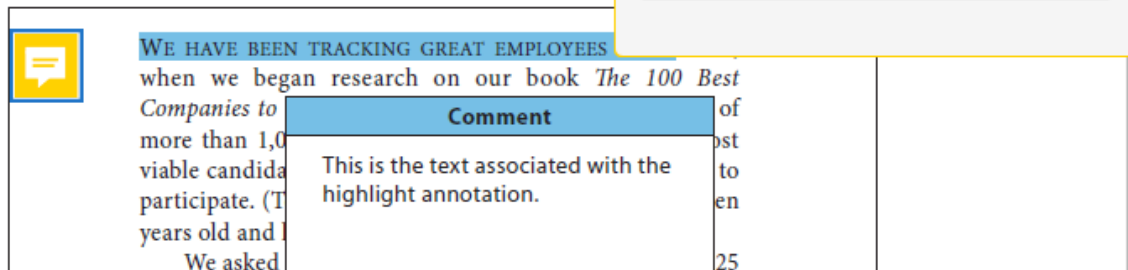
18.8 Polygon

If you want to add a polygon like this:

you can use *Polygon*:

```
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Polygon

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the line
annotation = Polygon(
    vertices=[(50, 550), (200, 650), (70, 750), (50, 700)],
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

18.9 Popup

Manage the Popup windows for markups. looks like this:

you can use the *Popup*:

you have to use the returned result from add_annotation() to fill-up the

```python
from pypdf.annotations import Popup, Text

# Arrange
writer = pypdf.PdfWriter()
writer.append(os.path.join(RESOURCE_ROOT, "crazyones.pdf"), [0])

# Act
text_annotation = writer.add_annotation(
    0,
    Text(
        text="Hello World\nThis is the second line!",
        rect=(50, 550, 200, 650),
        open=True,
    ),
)

popup_annotation = Popup(
    rect=(50, 550, 200, 650),
    open=True,
    parent=text_annotation,  # use the output of add_annotation
)

writer.write("annotated-pdf-popup.pdf")
```

18.10 Link

If you want to add a link, you can use *Link*:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Link

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the line
annotation = Link(
    rect=(50, 550, 200, 650),
    url="https://martin-thoma.com/",
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

You can also add internal links:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Link

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)

# Add the line
annotation = Link(
    rect=(50, 550, 200, 650), target_page_index=3, fit="/FitH", fit_args=(123,)
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

18.11 Text Markup Annotations

Text markup annotations refer to a specific piece of text within the document.

Those are a bit more complicated as you need to know exactly where the text is. Those are the "Quad points".

18.11.1 Highlighting

If you want to highlight text like this:

you can use the *Highlight*:

```python
from pypdf import PdfReader, PdfWriter
from pypdf.annotations import Highlight

pdf_path = os.path.join(RESOURCE_ROOT, "crazyones.pdf")
reader = PdfReader(pdf_path)
page = reader.pages[0]
writer = PdfWriter()
writer.add_page(page)
```

(continues on next page)

```python
# Add the highlight
annotation = Highlight(
    vertices=[(50, 550), (200, 650), (70, 750), (50, 700)],
)
writer.add_annotation(page_number=0, annotation=annotation)

# Write the annotated file to disk
with open("annotated-pdf.pdf", "wb") as fp:
    writer.write(fp)
```

ADDING VIEWER PREFERENCES

It is possible to set viewer preferences of the PDF file. These properties are described in Section 12.2 of the PDF 1.7 specification.

Note that the /ViewerPreferences dictionary does not exist by default. If it's not already present, it must be created by calling the create_viewer_preferences method of the PdfWriter object.

If viewer preferences exist in a PDF file being read with PdfReader, you can access them as properties of the viewer_preferences properties. Otherwise, the viewer_preferences property will be set to None.

19.1 Example

```python
from pypdf import PdfWriter
from pypdf.generic import ArrayObject, NumberObject

writer = PdfWriter()

writer.create_viewer_preferences()

# /HideToolbar
writer.viewer_preferences.hide_toolbar = True
# /HideMenubar
writer.viewer_preferences.hide_menubar = True
# /HideWindowUI
writer.viewer_preferences.hide_windowui = True
# /FitWindow
writer.viewer_preferences.fit_window = True
# /CenterWindow
writer.viewer_preferences.center_window = True
# /DisplayDocTitle
writer.viewer_preferences.display_doctitle = True


# /NonFullScreenPageMode
writer.viewer_preferences.non_fullscreen_pagemode = "/UseNone"  # default
writer.viewer_preferences.non_fullscreen_pagemode = "/UseOutlines"
writer.viewer_preferences.non_fullscreen_pagemode = "/UseThumbs"
writer.viewer_preferences.non_fullscreen_pagemode = "/UseOC"

# /Direction
writer.viewer_preferences.direction = "/L2R"  # default
```

(continues on next page)

```python
writer.viewer_preferences.direction = "/R2L"

# /ViewArea
writer.viewer_preferences.view_area = "/CropBox"
# /ViewClip
writer.viewer_preferences.view_clip = "/CropBox"
# /PrintArea
writer.viewer_preferences.print_area = "/CropBox"
# /PrintClip
writer.viewer_preferences.print_clip = "/CropBox"

# /PrintScaling
writer.viewer_preferences.print_scaling = "/None"
writer.viewer_preferences.print_scaling = "/AppDefault"  # default according to PDF spec

# /Duplex
writer.viewer_preferences.duplex = "/Simplex"
writer.viewer_preferences.duplex = "/DuplexFlipShortEdge"
writer.viewer_preferences.duplex = "/DuplexFlipLongEdge"

# /PickTrayByPDFSize
writer.viewer_preferences.pick_tray_by_pdfsize = True
# /PrintPageRange
writer.viewer_preferences.print_pagerange = ArrayObject(
    [NumberObject("1"), NumberObject("10"), NumberObject("20"), NumberObject("30")]
)
# /NumCopies
writer.viewer_preferences.num_copies = 2

for i in range(40):
    writer.add_blank_page(10, 10)

with open("output.pdf", "wb") as output_stream:
    writer.write(output_stream)
```

(The names beginning with a slash character are part of the PDF file format. They are included here to aid to anyone searching pypdf documentation for these names from the PDF specification.)

INTERACTIONS WITH PDF FORMS

20.1 Reading form fields

```python
from pypdf import PdfReader

reader = PdfReader("form.pdf")
fields = reader.get_form_text_fields()
fields == {"key": "value", "key2": "value2"}
```

20.2 Filling out forms

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("form.pdf")
writer = PdfWriter()

page = reader.pages[0]
fields = reader.get_fields()

writer.append(reader)

writer.update_page_form_field_values(
    writer.pages[0], {"fieldname": "some filled in text"}
)

# write "output" to pypdf-output.pdf
with open("filled-out.pdf", "wb") as output_stream:
    writer.write(output_stream)
```

STREAMING DATA WITH PYPDF

In some cases you might want to avoid saving things explicitly as a file to disk, e.g. when you want to store the PDF in a database or AWS S3.

pypdf supports streaming data to a file-like object and here is how.

```python
from io import BytesIO

# Prepare example
with open("example.pdf", "rb") as fh:
    bytes_stream = BytesIO(fh.read())

# Read from bytes_stream
reader = PdfReader(bytes_stream)

# Write to bytes_stream
writer = PdfWriter()
with BytesIO() as bytes_stream:
    writer.write(bytes_stream)
```

21.1 Writing a PDF directly to AWS S3

Suppose you want to manipulate a PDF and write it directly to AWS S3 without having to write the document to a file first. We have the original PDF in `raw_bytes_data` as `bytes` and want to set `my-secret-password`:

```python
from io import BytesIO

import boto3
from pypdf import PdfReader, PdfWriter


reader = PdfReader(BytesIO(raw_bytes_data))
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Add a password to the new PDF
writer.encrypt("my-secret-password")
```

(continues on next page)

```python
# Save the new PDF to a file
with BytesIO() as bytes_stream:
    writer.write(bytes_stream)
    bytes_stream.seek(0)
    s3 = boto3.client("s3")
    s3.write_get_object_response(
        Body=bytes_stream, RequestRoute=request_route, RequestToken=request_token
    )
```

21.2 Reading PDFs directly from cloud services

One option is to first download the file and then pass the local file path to `PdfReader`. Another option is to get a byte stream.

For AWS S3 it works like this:

```python
from io import BytesIO

import boto3
from pypdf import PdfReader


s3 = boto3.client("s3")
obj = s3.get_object(Body=csv_buffer.getvalue(), Bucket="my-bucket", Key="my/doc.pdf")
reader = PdfReader(BytesIO(obj["Body"].read()))
```

It works similarly for Google Cloud Storage (example).

REDUCE PDF FILE SIZE

There are multiple ways to reduce the size of a given PDF file. The easiest one is to remove content (e.g. images) or pages.

22.1 Removing duplication

Some PDF documents contain the same object multiple times. For example, if an image appears three times in a PDF it could be embedded three times. Or it can be embedded once and referenced twice.

This can be done by reading and writing the file:

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("big-old-file.pdf")
writer = PdfWriter()

for page in reader.pages:
    writer.add_page(page)

writer.add_metadata(reader.metadata)

with open("smaller-new-file.pdf", "wb") as fp:
    writer.write(fp)
```

It depends on the PDF how well this works, but we have seen an 86% file reduction (from 5.7 MB to 0.8 MB) within a real PDF.

22.2 Removing Images

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

for page in reader.pages:
    writer.add_page(page)

writer.remove_images()
```

```python
with open("out.pdf", "wb") as f:
    writer.write(f)
```

22.3 Reducing Image Quality

If we reduce the quality of the images within the PDF, we can **sometimes** reduce the file size of the PDF overall. That depends on how well the reduced quality image can be compressed.

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

for page in reader.pages:
    writer.add_page(page)

for page in writer.pages:
    for img in page.images:
        img.replace(img.image, quality=80)

with open("out.pdf", "wb") as f:
    writer.write(f)
```

22.4 Lossless Compression

pypdf supports the FlateDecode filter which uses the zlib/deflate compression method. It is a lossless compression, meaning the resulting PDF looks exactly the same.

Deflate compression can be applied to a page via `page.compress_content_streams`:

```python
from pypdf import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

for page in reader.pages:
    writer.add_page(page)

for page in writer.pages:
    # This has to be done on the writer, not the reader!
    page.compress_content_streams()  # This is CPU intensive!

with open("out.pdf", "wb") as f:
    writer.write(f)
```

`page.compress_content_streams` uses `zlib.compress` and supports the `level` parameter: `level=0` means no compression, `level=9` refers to the highest compression.

Using this method, we have seen a reduction by 70% (from 11.8 MB to 3.5 MB) with a real PDF.

22.5 Removing Sources

When a page is removed from the page list, its content will still be present in the PDF file. This means that the data may still be used elsewhere.

Simply removing a page from the page list will reduce the page count but not the file size. In order to exclude the content completely, the pages should not be added to the PDF using the PdfWriter.append() function. Instead, only the desired pages should be selected for inclusion (note: PR #1843 will add a page deletion feature).

There can be issues with poor PDF formatting, such as when all pages are linked to the same resource. In such cases, dropping references to specific pages becomes useless because there is only one source for all pages.

Cropping is an ineffective method for reducing the file size because it only adjusts the viewboxes and not the external parts of the source image. Therefore, the content that is no longer visible will still be present in the PDF.

PDF VERSION SUPPORT

PDF comes in the following versions:

- 1993: 1.0
- 1994: 1.1
- 1996: 1.2
- 1999: 1.3
- 2001: 1.4
- 2003: 1.5
- 2004: 1.6
- 2006 - 2012: 1.7, ISO 32000-1:2008
- 2017: 2.0

The general format didn't change, but new features got added. It can be that pypdf can do the operations you want on PDF 2.0 files without fully supporting all features of PDF 2.0.

23.1 PDF Feature Support by pypdf

Feature	PDF-Version	pypdf Support
Transparent Graphics	1.4	?
CMaps	1.4	
Object Streams	1.5	?
Cross-reference Streams	1.5	?
Optional Content Groups (OCGs) - Layers	1.5	?
Content Stream Compression	1.5	?
AES Encryption	1.6	

See History of PDF for more features.

Some PDF features are not supported by pypdf, but other libraries can be used for them:

- pyHanko: Cryptographically sign a PDF (#302)
- camelot-py: Table Extraction (#231)

PDF/A COMPLIANCE

PDF/A is a specialized, ISO-standardized version of the Portable Document Format (PDF) specifically designed for the long-term preservation and archiving of electronic documents. It ensures that files remain accessible, readable, and true to their original appearance by embedding all necessary fonts, images, and metadata within the document itself. By adhering to strict guidelines and minimizing dependencies on external resources or proprietary software, PDF/A ensures the consistent and reliable reproduction of content, safeguarding it against future technological changes and obsolescence.

24.1 PDF/A Versions

- **PDF/A-1**: Based on PDF 1.4, PDF/A-1 is the first version of the standard and is divided into two levels: PDF/A-1a (Level A, ensuring accessibility) and PDF/A-1b (Level B, ensuring visual preservation).

    - **Level B** (Basic): Ensures visual preservation and basic requirements for archiving.

    - **Level A** (Accessible): Everything from level B, but includes additional requirements for accessibility, such as tagging, Unicode character mapping, and logical structure.

- **PDF/A-2**: Based on PDF 1.7 (ISO 32000-1), PDF/A-2 adds features and improvements over PDF/A-1, while maintaining compatibility with PDF/A-1b (Level B) documents.

    - **Level B** (Basic): Like PDF/A-1b, but support for PDF 1.7 features such as transparency layers.

    - **Level U** (Unicode): Ensures Unicode mapping without the full accessibility requirements of PDF/A-1a (Level A).

    - **Level A** (Accessible): Similar to PDF/A-1a

- **PDF/A-3**: Based on PDF 1.7 (ISO 32000-1), PDF/A-3 is similar to PDF/A-2 but allows the embedding of non-PDF/A files as attachments, enabling the archiving of source or supplementary data alongside the PDF/A document. This is interesting for invoices which can add XML files.

- **PDF/A-4**: Based on PDF 2.0 (ISO 32000-2), PDF/A-4 introduces new features and improvements for better archiving and accessibility. The previous levels are replaced by PDF/A-4f (ensuring visual preservation and allowing attachments) and PDF/A-4e (Engineering, allows 3D content).

24.2 PDF/A-1b

In contrast to other PDF documents, PDF/A-1b documents must fulfill those requirements:

- **MarkInfo Object**: The MarkInfo object is a dictionary object within a PDF/A file that provides information about the logical structure and tagging of the document. The MarkInfo object indicates whether the document is tagged, contains optional content, or has a structure tree that describes the logical arrangement of content such as headings, paragraphs, lists, and tables. By including the MarkInfo object, PDF/A ensures that electronic documents are accessible to users with disabilities, such as those using screen readers or other assistive technologies.

- **Embedded fonts**: All fonts used in the document must be embedded to ensure consistent text rendering across different devices and systems.

- **Color Spaces**: DeviceRGB is a device-dependent color space that relies on the specific characteristics of the output device, which can lead to inconsistent color rendering across various devices. To achieve accurate and consistent color representation, PDF/A requires the use of device-independent color spaces, such as ICC-based color profiles.

- **XMP (Extensible Metadata Platform) metadata**: XMP metadata provides a standardized and extensible way to store essential information about a document and its properties. XMP metadata is an XML-based format embedded directly within a PDF/A file. It contains various types of information, such as document title, author, creation and modification dates, keywords, and copyright information, as well as PDF/A-specific details like conformance level and OutputIntent.

24.3 Validation

VeraPDF is the go-to PDF/A validator.

There are several online-validators which allow you to simply upload the document:

- pdfen.com

- avepdf.com : Gives an error report

- pdfa.org

- visual-paradigm.com - can convert the PDF to a PDF/A

- pdf2go.com

- slub-dresden.de links to relevant parts in the specification.

24.4 pypdf and PDF/A

At the moment, pypdf does not make any guarantees regarding PDF/A. Support is very welcome.

THE PDFREADER CLASS

class pypdf.**PdfReader**(*stream: Union[str, IO[Any], Path]*, *strict: bool = False*, *password: Union[None, str, bytes] = None*)

Bases: object

Initialize a PdfReader object.

This operation can take some time, as the PDF stream's cross-reference tables are read into memory.

>   **Parameters**
>
>   - **stream** – A File object or an object that supports the standard read and seek methods similar to a File object. Could also be a string representing a path to a PDF file.
>
>   - **strict** – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to False.
>
>   - **password** – Decrypt PDF file at initialization. If the password is None, the file will not be decrypted. Defaults to None

property viewer_preferences: Optional[pypdf.generic._viewerpref.ViewerPreferences]

>   Returns the existing ViewerPreferences as an overloaded dictionary.

property pdf_header: str

>   The first 8 bytes of the file.

>   This is typically something like '%PDF-1.6' and can be used to detect if the file is actually a PDF file and which version it is.

property metadata: Optional[*DocumentInformation*]

>   Retrieve the PDF file's document information dictionary, if it exists.

>   Note that some PDF files use metadata streams instead of docinfo dictionaries, and these metadata streams will not be accessed by this function.

getDocumentInfo() → Optional[*DocumentInformation*]

>   Use the attribute *metadata* instead.

>   Deprecated since version 1.28.0.

property documentInfo: Optional[*DocumentInformation*]

>   Use the attribute *metadata* instead.

>   Deprecated since version 1.28.0.

property xmp_metadata: Optional[*XmpInformation*]

>   XMP (Extensible Metadata Platform) data.

getXmpMetadata() → Optional[*XmpInformation*]

    Use the attribute *metadata* instead.

    Deprecated since version 1.28.0.

property xmpMetadata: Optional[_XmpInformation_**]**

    Use the attribute *xmp_metadata* instead.

    Deprecated since version 1.28.0..

getNumPages() → int

    Use len(reader.pages) instead.

    Deprecated since version 1.28.0.

property numPages: int

    Use len(reader.pages) instead.

    Deprecated since version 1.28.0.

getPage(*pageNumber: int*) → *PageObject*

    Use reader.pages[page_number] instead.

    Deprecated since version 1.28.0.

property namedDestinations: Dict[str, Any]

    Use *named_destinations* instead.

    Deprecated since version 1.28.0.

property named_destinations: Dict[str, Any]

    A read-only dictionary which maps names to *Destinations*

get_fields(*tree: Optional[TreeObject] = None*, *retval: Optional[Dict[Any, Any]] = None*, *fileobj: Optional[Any] = None*) → Optional[Dict[str, Any]]

    Extract field data if this PDF contains interactive form fields.

    The *tree* and *retval* parameters are for recursive use.

        Parameters

            • **tree** –

            • **retval** –

            • **fileobj** – A file object (usually a text file) to write a report to on all interactive form fields found.

        Returns A dictionary where each key is a field name, and each value is a *Field* object. By default, the mapping name is used for keys. None if form data could not be located.

getFields(*tree: Optional[TreeObject] = None*, *retval: Optional[Dict[Any, Any]] = None*, *fileobj: Optional[Any] = None*) → Optional[Dict[str, Any]]

    Use *get_fields()* instead.

    Deprecated since version 1.28.0.

get_form_text_fields(*full_qualified_name: bool = False*) → Dict[str, Any]

    Retrieve form fields from the document with textual data.

        Parameters full_qualified_name – to get full name

Returns

A dictionary. The key is the name of the form field, the value is the content of the field.

If the document contains multiple form fields with the same name, the second and following will get the suffix .2, .3, …

getFormTextFields() → Dict[str, Any]

Use *get_form_text_fields()* instead.

Deprecated since version 1.28.0.

getNamedDestinations(*tree: Optional[TreeObject] = None*, *retval: Optional[Any] = None*) → Dict[str, Any]

Use *named_destinations* instead.

Deprecated since version 1.28.0.

property outline: List[Union[*Destination*, List[Union[*Destination*, List[*Destination*]]]]]

Read-only property for the outline present in the document.

(i.e., a collection of 'outline items' which are also known as 'bookmarks')

property outlines: List[Union[*Destination*, List[Union[*Destination*, List[*Destination*]]]]]

Use *outline* instead.

Deprecated since version 2.9.0.

getOutlines(*node: Optional[DictionaryObject] = None*, *outline: Optional[Any] = None*) → List[Union[*Destination*, List[Union[*Destination*, List[*Destination*]]]]]

Use *outline* instead.

Deprecated since version 1.28.0.

property threads: Optional[pypdf.generic._data_structures.ArrayObject]

Read-only property for the list of threads.

See §8.3.2 from PDF 1.7 spec.

It's an array of dictionaries with "/F" and "/I" properties or None if there are no articles.

get_page_number(*page:* PageObject) → int

Retrieve page number of a given PageObject.

Parameters page – The page to get page number. Should be an instance of *PageObject*

Returns The page number or -1 if page is not found

getPageNumber(*page:* PageObject) → int

Use *get_page_number()* instead.

Deprecated since version 1.28.0.

get_destination_page_number(*destination:* Destination) → int

Retrieve page number of a given Destination object.

Parameters destination – The destination to get page number.

Returns The page number or -1 if page is not found

getDestinationPageNumber(*destination:* Destination) → int

    Use *get_destination_page_number()* instead.

    Deprecated since version 1.28.0.

property pages: List[*PageObject*]

    Read-only property that emulates a list of Page objects.

property page_labels: List[str]

    A list of labels for the pages in this document.

    This property is read-only. The labels are in the order that the pages appear in the document.

property page_layout: Optional[str]

    Get the page layout currently being used.

Table 1: Valid `layout` values

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

getPageLayout() → Optional[str]

    Use *page_layout* instead.

    Deprecated since version 1.28.0.

property pageLayout: Optional[str]

    Use *page_layout* instead.

    Deprecated since version 1.28.0.

property page_mode: Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

    Get the page mode currently being used.

Table 2: Valid `mode` values

/UseNone	Do not show outline or thumbnails panels
/UseOutlines	Show outline (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

getPageMode() → Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

    Use *page_mode* instead.

    Deprecated since version 1.28.0.

property pageMode: Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

> Use *page_mode* instead.
>
> Deprecated since version 1.28.0.

get_object(*indirect_reference: Union[int, IndirectObject]*) → Optional[PdfObject]

getObject(*indirectReference: IndirectObject*) → Optional[PdfObject]

> Use *get_object()* instead.
>
> Deprecated since version 1.28.0.

read_object_header(*stream: IO[Any]*) → Tuple[int, int]

readObjectHeader(*stream: IO[Any]*) → Tuple[int, int]

> Use *read_object_header()* instead.
>
> Deprecated since version 1.28.0.

cache_get_indirect_object(*generation: int, idnum: int*) → Optional[PdfObject]

cacheGetIndirectObject(*generation: int, idnum: int*) → Optional[PdfObject]

> Use *cache_get_indirect_object()* instead.
>
> Deprecated since version 1.28.0.

cache_indirect_object(*generation: int, idnum: int, obj: Optional[PdfObject]*) → Optional[PdfObject]

cacheIndirectObject(*generation: int, idnum: int, obj: Optional[PdfObject]*) → Optional[PdfObject]

> Use *cache_indirect_object()* instead.
>
> Deprecated since version 1.28.0.

read(*stream: IO[Any]*) → None

read_next_end_line(*stream: IO[Any], limit_offset: int = 0*) → bytes

> Deprecated since version 2.1.0.

readNextEndLine(*stream: IO[Any], limit_offset: int = 0*) → bytes

> Deprecated since version 1.28.0.

decrypt(*password: Union[str, bytes]*) → PasswordType

> When using an encrypted / secured PDF file with the PDF Standard encryption handler, this function will allow the file to be decrypted. It checks the given password against the document's user password and owner password, and then stores the resulting decryption key if either password is correct.
>
> It does not matter which password was matched. Both passwords provide the correct decryption key that will allow the document to be used with this library.
>
> > **Parameters password** – The password to match.
> >
> > **Returns** An indicator if the document was decrypted and weather it was the owner password or the user password.

decode_permissions(*permissions_code: int*) → Dict[str, bool]

property is_encrypted: bool

> Read-only boolean property showing whether this PDF file is encrypted.
>
> Note that this property, if true, will remain true even after the *decrypt()* method is called.

getIsEncrypted() → bool

   Use *is_encrypted* instead.

   Deprecated since version 1.28.0.

property isEncrypted: bool

   Use *is_encrypted* instead.

   Deprecated since version 1.28.0.

property xfa: Optional[Dict[str, Any]]

add_form_topname(*name: str*) → Optional[DictionaryObject]

   Add a top level form that groups all form fields below it.

   Parameters name – text string of the "/T" Attribute of the created object

   Returns The created object. None means no object was created.

rename_form_topname(*name: str*) → Optional[DictionaryObject]

   Rename top level form field that all form fields below it.

   Parameters name – text string of the "/T" field of the created object

   Returns The modified object. None means no object was modified.

property attachments: Mapping[str, List[bytes]]

THE PDFWRITER CLASS

class pypdf.**PdfWriter**(*fileobj: Union[str, IO[Any]] = ''*, *clone_from: Union[None, PdfReader, str, IO[Any], Path] = None*)

> Bases: object
>
> Write a PDF file out, given pages produced by another class.
>
> Typically data is added from a *PdfReader*.
>
> **property pdf_header: bytes**
>> Header of the PDF document that is written.
>>
>> This should be something like b'%PDF-1.5'. It is recommended to set the lowest version that supports all features which are used within the PDF file.
>
> **get_object**(*indirect_reference: Union[None, int, IndirectObject] = None*, *ido: Optional[IndirectObject] = None*) → PdfObject
>
> **getObject**(*ido: Union[int, IndirectObject]*) → PdfObject
>> Use *get_object()* instead.
>>
>> Deprecated since version 1.28.0.
>
> **set_need_appearances_writer**(*state: bool = True*) → None
>> Sets the "NeedAppearances" flag in the PDF writer.
>>
>> The "NeedAppearances" flag indicates whether the appearance dictionary for form fields should be automatically generated by the PDF viewer or if the embedded appearence should be used.
>>
>>> **Parameters state** – The actual value of the NeedAppearances flag.
>>>
>>> **Returns** None
>
> **property viewer_preferences: Optional[pypdf.generic._viewerpref.ViewerPreferences]**
>> Returns the existing ViewerPreferences as an overloaded dictionary.
>
> **create_viewer_preferences**() → ViewerPreferences
>
> **add_page**(*page: PageObject*, *excluded_keys: Iterable[str] = ()*) → *PageObject*
>> Add a page to this PDF file.
>>
>> Recommended for advanced usage including the adequate excluded_keys.
>>
>> The page is usually acquired from a *PdfReader* instance.
>>
>>> **Parameters**
>>>
>>> • **page** – The page to add to the document. Should be an instance of *PageObject*

- **excluded_keys** –

> **Returns** The added PageObject.

addPage(*page:* PageObject, *excluded_keys:* *Iterable[str]* = ()) → *PageObject*

Use *add_page()* instead.

Deprecated since version 1.28.0..

insert_page(*page:* PageObject, *index:* *int* = 0, *excluded_keys:* *Iterable[str]* = ()) → *PageObject*

Insert a page in this PDF file. The page is usually acquired from a *PdfReader* instance.

> **Parameters**
>
> - **page** – The page to add to the document.
>
> - **index** – Position at which the page will be inserted.
>
> - **excluded_keys** –
>
> **Returns** The added PageObject.

insertPage(*page:* PageObject, *index:* *int* = 0, *excluded_keys:* *Iterable[str]* = ()) → *PageObject*

Use *insert_page()* instead.

Deprecated since version 1.28.0.

get_page(*page_number:* *Optional[int]* = None, *pageNumber:* *Optional[int]* = None) → *PageObject*

Retrieve a page by number from this PDF file.

> **Parameters** **page_number** – The page number to retrieve (pages begin at zero)
>
> **Returns** The page at the index given by *page_number*

getPage(*pageNumber:* *int*) → *PageObject*

Use `writer.pages[page_number]` instead.

Deprecated since version 1.28.0.

getNumPages() → int

Use `len(writer.pages)` instead.

Deprecated since version 1.28.0.

property pages: List[PageObject**]**

Property that emulates a list of *PageObject*. this property allows to get a page or a range of pages.

It provides also capability to remove a page/range of page from the list (through del operator) Note: only the page entry is removed. As the objects beneath can be used somewhere else. a solution to completely remove them - if they are not used somewhere - is to write to a buffer/temporary and to then load it into a new PdfWriter object.

add_blank_page(*width:* *Optional[float]* = None, *height:* *Optional[float]* = None) → *PageObject*

Append a blank page to this PDF file and returns it.

If no page size is specified, use the size of the last page.

> **Parameters**
>
> - **width** – The width of the new page expressed in default user space units.
>
> - **height** – The height of the new page expressed in default user space units.
>
> **Returns** The newly appended page

> Raises **PageSizeNotDefinedError** – if width and height are not defined and previous page does not exist.

addBlankPage(*width: Optional[float] = None*, *height: Optional[float] = None*) → *PageObject*

Use *add_blank_page()* instead.

Deprecated since version 1.28.0.

insert_blank_page(*width: Optional[Union[float, Decimal]] = None*, *height: Optional[Union[float, Decimal]] = None*, *index: int = 0*) → *PageObject*

Insert a blank page to this PDF file and returns it.

If no page size is specified, use the size of the last page.

> **Parameters**
>
> - **width** – The width of the new page expressed in default user space units.
>
> - **height** – The height of the new page expressed in default user space units.
>
> - **index** – Position to add the page.
>
> **Returns** The newly appended page
>
> Raises **PageSizeNotDefinedError** – if width and height are not defined and previous page does not exist.

insertBlankPage(*width: Optional[Union[float, Decimal]] = None*, *height: Optional[Union[float, Decimal]] = None*, *index: int = 0*) → *PageObject*

Use *insertBlankPage()* instead.

Deprecated since version 1.28.0..

property open_destination: **Union[**None**,** *Destination***,**
pypdf.generic._base.TextStringObject, pypdf.generic._base.ByteStringObject]

Property to access the opening destination (/OpenAction entry in the PDF catalog). It returns None if the entry does not exist is not set.

> Raises **Exception** – If a destination is invalid.

add_js(*javascript: str*) → None

Add Javascript which will launch upon opening this PDF.

> **Parameters javascript** – Your Javascript.

```
>>> output.add_js("this.print({bUI:true,bSilent:false,bShrinkToFit:true});")
# Example: This will launch the print window when the PDF is opened.
```

addJS(*javascript: str*) → None

Use *add_js()* instead.

Deprecated since version 1.28.0.

add_attachment(*filename: str*, *data: Union[str, bytes]*) → None

Embed a file inside the PDF.

Reference: https://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf
Section 7.11.3

> **Parameters**
>
> - **filename** – The filename to display.

- **data** – The data in the file.

addAttachment(*fname: str, fdata: Union[str, bytes]*) → None

Use `add_attachment()` instead.

Deprecated since version 1.28.0.

append_pages_from_reader(*reader: PdfReader, after_page_append: Optional[Callable[[PageObject], None]] = None*) → None

Copy pages from reader to writer. Includes an optional callback parameter which is invoked after pages are appended to the writer.

append should be prefered.

> **Parameters**
>
> - **reader** – a PdfReader object from which to copy page annotations to this writer object. The writer's annots will then be updated
>
> - **after_page_append** – Callback function that is invoked after each page is appended to the writer. Signature includes a reference to the appended page (delegates to append_pages_from_reader). The single parameter of the callback is a reference to the page just appended to the document.

appendPagesFromReader(*reader: PdfReader, after_page_append: Optional[Callable[[PageObject], None]] = None*) → None

Use `append_pages_from_reader()` instead.

Deprecated since version 1.28.0.

update_page_form_field_values(*page: ~pypdf._page.PageObject, fields: ~typing.Dict[str, ~typing.Any], flags: ~pypdf.constants.FieldFlag = FieldFlag.None, auto_regenerate: ~typing.Optional[bool] = True*) → None

Update the form field values for a given page from a fields dictionary.

Copy field texts and values from fields to page. If the field links to a parent object, add the information to the parent.

> **Parameters**
>
> - **page** – Page reference from PDF writer where the annotations and field data will be updated.
>
> - **fields** – a Python dictionary of field names (/T) and text values (/V)
>
> - **flags** – An integer (0 to 7). The first bit sets ReadOnly, the second bit sets Required, the third bit sets NoExport. See PDF Reference Table 8.70 for details.
>
> - **auto_regenerate** – set/unset the need_appearances flag ; the flag is unchanged if auto_regenerate is None

updatePageFormFieldValues(*page: ~pypdf._page.PageObject, fields: ~typing.Dict[str, ~typing.Any], flags: ~pypdf.constants.FieldFlag = FieldFlag.None*) → None

Use `update_page_form_field_values()` instead.

Deprecated since version 1.28.0.

clone_reader_document_root(*reader: PdfReader*) → None

Copy the reader document root to the writer and all sub elements, including pages, threads, outlines,. . . For partial insertion, append should be considered.

> **Parameters reader** – PdfReader from the document root should be copied.

cloneReaderDocumentRoot(*reader:* PdfReader) → None

> Use *clone_reader_document_root()* instead.
>
> Deprecated since version 1.28.0.

clone_document_from_reader(*reader:* PdfReader, *after_page_append: Optional[Callable[[PageObject], None]] = None*) → None

> Create a copy (clone) of a document from a PDF file reader cloning section '/Root' and '/Info' and '/ID' of the pdf.
>
> **Parameters**
>
> - **reader** – PDF file reader instance from which the clone should be created.
>
> - **after_page_append** – Callback function that is invoked after each page is appended to the writer. Signature includes a reference to the appended page (delegates to append_pages_from_reader). The single parameter of the callback is a reference to the page just appended to the document.

cloneDocumentFromReader(*reader:* PdfReader, *after_page_append: Optional[Callable[[PageObject], None]] = None*) → None

> Use *clone_document_from_reader()* instead.
>
> Deprecated since version 1.28.0.

generate_file_identifiers() → None

> Generate an identifier for the PDF that will be written.
>
> The only point of this is ensuring uniqueness. Reproducibility is not required; see 14.4 "File Identifiers".

encrypt(*user_password: Optional[str] = None, owner_password: Optional[str] = None, use_128bit: bool = True, permissions_flag: UserAccessPermissions = UserAccessPermissions.PRINT | MODIFY | EXTRACT | ADD_OR_MODIFY | R7 | R8 | FILL_FORM_FIELDS | EXTRACT_TEXT_AND_GRAPHICS | ASSEMBLE_DOC | PRINT_TO_REPRESENTATION | R13 | R14 | R15 | R16 | R17 | R18 | R19 | R20 | R21 | R22 | R23 | R24 | R25 | R26 | R27 | R28 | R29 | R30 | R31, user_pwd: Optional[str] = None, owner_pwd: Optional[str] = None, *, algorithm: Optional[str] = None*) → None

> Encrypt this PDF file with the PDF Standard encryption handler.
>
> **Parameters**
>
> - **user_password** – The password which allows for opening and reading the PDF file with the restrictions provided.
>
> - **owner_password** – The password which allows for opening the PDF files without any restrictions. By default, the owner password is the same as the user password.
>
> - **use_128bit** – flag as to whether to use 128bit encryption. When false, 40bit encryption will be used. By default, this flag is on.
>
> - **permissions_flag** – permissions as described in TABLE 3.20 of the PDF 1.7 specification. A bit value of 1 means the permission is grantend. Hence an integer value of -1 will set all flags. Bit position 3 is for printing, 4 is for modifying content, 5 and 6 control annotations, 9 for form fields, 10 for extraction of text and graphics.
>
> - **algorithm** – encrypt algorithm. Values maybe one of "RC4-40", "RC4-128", "AES-128", "AES-256-R5", "AES-256". If it's valid, *use_128bit* will be ignored.

write_stream(*stream: IO[Any]*) → None

write(*stream: [Union](https://docs.python.org/3/library/typing.html#typing.Union)[[Path](https://docs.python.org/3/library/pathlib.html#pathlib.Path), [str](https://docs.python.org/3/library/stdtypes.html#str), [IO](https://docs.python.org/3/library/typing.html#typing.IO)[[Any](https://docs.python.org/3/library/typing.html#typing.Any)]]*) → [Tuple](https://docs.python.org/3/library/typing.html#typing.Tuple)[[bool](https://docs.python.org/3/library/functions.html#bool), [IO](https://docs.python.org/3/library/typing.html#typing.IO)]

> Write the collection of pages added to this object out as a PDF file.
>
> > **Parameters** `stream` – An object to write the file to. The object can support the write method and the tell method, similar to a file object, or be a file path, just like the fileobj, just named it stream to keep existing workflow.
> >
> > **Returns** A tuple (bool, IO)

add_metadata(*infos: [Dict](https://docs.python.org/3/library/typing.html#typing.Dict)[[str](https://docs.python.org/3/library/stdtypes.html#str), [Any](https://docs.python.org/3/library/typing.html#typing.Any)]*) → [None](https://docs.python.org/3/library/constants.html#None)

> Add custom metadata to the output.
>
> > **Parameters** `infos` – a Python dictionary where each key is a field and each value is your new metadata.

addMetadata(*infos: [Dict](https://docs.python.org/3/library/typing.html#typing.Dict)[[str](https://docs.python.org/3/library/stdtypes.html#str), [Any](https://docs.python.org/3/library/typing.html#typing.Any)]*) → [None](https://docs.python.org/3/library/constants.html#None)

> Use [add_metadata()](#) instead.
>
> Deprecated since version 1.28.0.

get_reference(*obj: PdfObject*) → IndirectObject

getReference(*obj: PdfObject*) → IndirectObject

> Use [get_reference()](#) instead.
>
> Deprecated since version 1.28.0.

get_outline_root() → TreeObject

get_threads_root() → ArrayObject

> The list of threads.
>
> See §8.3.2 from PDF 1.7 spec.
>
> > **Returns** An array (possibly empty) of Dictionaries with /F and /I properties.

property threads: `pypdf.generic._data_structures.ArrayObject`

> Read-only property for the list of threads.
>
> See §8.3.2 from PDF 1.7 spec.
>
> Each element is a dictionaries with /F and /I keys.

getOutlineRoot() → TreeObject

> Use [get_outline_root()](#) instead.
>
> Deprecated since version 1.28.0.

get_named_dest_root() → ArrayObject

getNamedDestRoot() → ArrayObject

> Use [get_named_dest_root()](#) instead.
>
> Deprecated since version 1.28.0.

add_outline_item_destination(*page_destination: [Union](https://docs.python.org/3/library/typing.html#typing.Union)[[None](https://docs.python.org/3/library/constants.html#None), IndirectObject, [PageObject](#), TreeObject] = None, parent: [Union](https://docs.python.org/3/library/typing.html#typing.Union)[[None](https://docs.python.org/3/library/constants.html#None), TreeObject, IndirectObject] = None, before: [Union](https://docs.python.org/3/library/typing.html#typing.Union)[[None](https://docs.python.org/3/library/constants.html#None), TreeObject, IndirectObject] = None, is_open: [bool](https://docs.python.org/3/library/functions.html#bool) = True, dest: [Union](https://docs.python.org/3/library/typing.html#typing.Union)[[None](https://docs.python.org/3/library/constants.html#None), [PageObject](#), TreeObject] = None*) → IndirectObject

add_bookmark_destination(*dest:* *Union*[PageObject, *TreeObject*]*, parent:* *Union*[*None, TreeObject, IndirectObject*] = *None*) → IndirectObject

    Use *add_outline_item_destination()* instead.

    Deprecated since version 2.9.0.

addBookmarkDestination(*dest:* PageObject*, parent:* *Optional*[*TreeObject*] = *None*) → IndirectObject

    Use *add_outline_item_destination()* instead.

    Deprecated since version 1.28.0.

add_outline_item_dict(*outline_item:* *Union*[*OutlineItem,* Destination*], parent:* *Union*[*None, TreeObject, IndirectObject*] = *None, before:* *Union*[*None, TreeObject, IndirectObject*] = *None, is_open:* *bool* = *True*) → IndirectObject

add_bookmark_dict(*outline_item:* *Union*[*OutlineItem,* Destination*], parent:* *Optional*[*TreeObject*] = *None*) → IndirectObject

    Use *add_outline_item_dict()* instead.

    Deprecated since version 2.9.0.

addBookmarkDict(*outline_item:* *Union*[*OutlineItem,* Destination*], parent:* *Optional*[*TreeObject*] = *None*) → IndirectObject

    Use *add_outline_item_dict()* instead.

    Deprecated since version 1.28.0.

add_outline_item(*title: str, page_number: ~typing.Union[None, ~pypdf._page.PageObject, ~pypdf.generic._base.IndirectObject, int], parent: ~typing.Union[None, ~pypdf.generic._data_structures.TreeObject, ~pypdf.generic._base.IndirectObject] = None, before: ~typing.Union[None, ~pypdf.generic._data_structures.TreeObject, ~pypdf.generic._base.IndirectObject] = None, color: ~typing.Optional[~typing.Union[~typing.Tuple[float, float, float], str]] = None, bold: bool = False, italic: bool = False, fit: ~pypdf.generic._fit.Fit = <pypdf.generic._fit.Fit object>, is_open: bool = True, pagenum: ~typing.Optional[int] = None*) → IndirectObject

    Add an outline item (commonly referred to as a "Bookmark") to the PDF file.

        Parameters

- **title** – Title to use for this outline item.

- **page_number** – Page number this outline item will point to.

- **parent** – A reference to a parent outline item to create nested outline items.

- **before** –

- **color** – Color of the outline item's font as a red, green, blue tuple from 0.0 to 1.0 or as a Hex String (#RRGGBB)

- **bold** – Outline item font is bold

- **italic** – Outline item font is italic

- **fit** – The fit of the destination page.

        Returns The added outline item as an indirect object.

add_bookmark(*title:* str*, pagenum:* int*, parent:* *Union*[*None, TreeObject, IndirectObject*] = *None, color:* *Optional*[*Tuple*[float, float, float]] = *None, bold:* bool = *False, italic:* bool = *False, fit:* Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit', *args:* *Union*[*NumberObject, NullObject,* float]) → IndirectObject

Use *add_outline_item()* instead.

Deprecated since version 2.9.0.

addBookmark(*title:* *str*, *pagenum:* *int*, *parent:* *Union[None, TreeObject, IndirectObject] = None*, *color:*
Optional[Tuple[float, float, float]] = None, *bold:* *bool = False*, *italic:* *bool = False*, *fit:*
Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit', **args:*
Union[NumberObject, NullObject, float]) → IndirectObject

Use *add_outline_item()* instead.

Deprecated since version 1.28.0.

add_outline() → None

add_named_destination_array(*title: TextStringObject*, *destination:* *Union[IndirectObject, ArrayObject]*)
→ None

add_named_destination_object(*page_destination:* *Optional[PdfObject] = None*, *dest:*
Optional[PdfObject] = None) → IndirectObject

addNamedDestinationObject(*dest:* Destination) → IndirectObject

Use *add_named_destination_object()* instead.

Deprecated since version 1.28.0.

add_named_destination(*title:* *str*, *page_number:* *Optional[int] = None*, *pagenum:* *Optional[int] = None*)
→ IndirectObject

addNamedDestination(*title:* *str*, *pagenum:* *int*) → IndirectObject

Use *add_named_destination()* instead.

Deprecated since version 1.28.0.

remove_links() → None

Remove links and annotations from this output.

removeLinks() → None

Use *remove_links()* instead.

Deprecated since version 1.28.0.

remove_annotations(*subtypes:* *Optional[Union[Literal['/Text', '/Link', '/FreeText', '/Line', '/Square',*
'/Circle', '/Polygon', '/PolyLine', '/Highlight', '/Unterline', '/Squiggly', '/StrikeOut',
'/Stamp', '/Caret', '/Ink', '/Popup', '/FileAttachment', '/Sound', '/Movie', '/Widget',
'/Screen', '/PrinterMark', '/TrapNet', '/Watermark', '/3D', '/Redact'],
Iterable[Literal['/Text', '/Link', '/FreeText', '/Line', '/Square', '/Circle', '/Polygon',
'/PolyLine', '/Highlight', '/Unterline', '/Squiggly', '/StrikeOut', '/Stamp', '/Caret', '/Ink',
'/Popup', '/FileAttachment', '/Sound', '/Movie', '/Widget', '/Screen', '/PrinterMark',
'/TrapNet', '/Watermark', '/3D', '/Redact']]]]) → None

Remove annotations by annotation subtype.

> **Parameters** **subtypes** – SubType or list of SubTypes to be removed. Examples are: "/Link",
> "/FileAttachment", "/Sound", "/Movie", "/Screen", ... If you want to remove all annotations,
> use subtypes=None.

remove_objects_from_page(*page:* *Union[PageObject, DictionaryObject]*, *to_delete:*
Union[ObjectDeletionFlag, Iterable[ObjectDeletionFlag]]) → None

Remove objects specified by `to_delete` from the given page.

Parameters

- **page** – Page object to clean up.

- **to_delete** – Objects to be deleted; can be a ObjectDeletionFlag or a list of Object-DeletionFlag

remove_images(*to_delete: ImageType = ImageType.ALL*, *ignore_byte_string_object: Optional[bool] = None*) → None

Remove images from this output.

Parameters

- **to_delete** – The type of images to be deleted (default = all images types)

- **ignore_byte_string_object** – deprecated

removeImages(*ignoreByteStringObject: bool = False*) → None

Use *remove_images()* instead.

Deprecated since version 1.28.0.

remove_text(*ignore_byte_string_object: Optional[bool] = None*) → None

Remove text from this output.

Parameters **ignore_byte_string_object** – deprecated

removeText(*ignoreByteStringObject: bool = False*) → None

Use *remove_text()* instead.

Deprecated since version 1.28.0.

add_uri(*page_number: int*, *uri: str*, *rect: RectangleObject*, *border: Optional[ArrayObject] = None*, *pagenum: Optional[int] = None*) → None

Add an URI from a rectangular area to the specified page.

This uses the basic structure of *add_link()*

Parameters

- **page_number** – index of the page on which to place the URI action.

- **uri** – URI of resource to link to.

- **rect** – *RectangleObject* or array of four integers specifying the clickable rectangular area [xLL, yLL, xUR, yUR], or string in the form "[xLL yLL xUR yUR]".

- **border** – if provided, an array describing border-drawing properties. See the PDF spec for details. No border will be drawn if this argument is omitted.

addURI(*pagenum: int*, *uri: str*, *rect: RectangleObject*, *border: Optional[ArrayObject] = None*) → None

Use *add_uri()* instead.

Deprecated since version 1.28.0.

add_link(*pagenum: int*, *page_destination: int*, *rect: RectangleObject*, *border: Optional[ArrayObject] = None*, *fit: Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit'*, **args: Union[NumberObject, NullObject, float]*) → DictionaryObject

addLink(*pagenum: int*, *page_destination: int*, *rect: RectangleObject*, *border: Optional[ArrayObject] = None*, *fit: Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit'*, **args: Union[NumberObject, NullObject, float]*) → None

Use *add_link()* instead.

Deprecated since version 1.28.0.

getPageLayout() → Optional[Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']]

Use *page_layout* instead.

Deprecated since version 1.28.0.

set_page_layout(*layout: Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']*) → None

Set the page layout.

> **Parameters** **layout** – The page layout to be used

Table 1: Valid `layout` arguments

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

setPageLayout(*layout: Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']*) → None

Use *page_layout* instead.

Deprecated since version 1.28.0.

property page_layout: **Optional[Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']]**

Page layout property.

Table 2: Valid `layout` values

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

property pageLayout: **Optional[Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']]**

Use *page_layout* instead.

Deprecated since version 1.28.0.

getPageMode() → Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

  Use *page_mode* instead.

  Deprecated since version 1.28.0.

set_page_mode(*mode: Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']*) → None

  Use *page_mode* instead.

  Deprecated since version 1.28.0.

setPageMode(*mode: Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']*) → None

  Use *page_mode* instead.

  Deprecated since version 1.28.0.

property page_mode: Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

  Page mode property.

Table 3: Valid mode values

/UseNone	Do not show outline or thumbnails panels
/UseOutlines	Show outline (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

property pageMode: Optional[Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']]

  Use *page_mode* instead.

  Deprecated since version 1.28.0.

add_annotation(*page_number: Union[int, PageObject]*, *annotation: Dict[str, Any]*) → DictionaryObject

  Add a single annotation to the page. The added annotation must be a new annotation. It can not be recycled.

   Parameters

    • **page_number** – PageObject or page index.

    • **annotation** – Annotation to be added (created with annotation).

   Returns The inserted object This can be used for pop-up creation, for example

clean_page(*page: Union[PageObject, IndirectObject]*) → *PageObject*

  Perform some clean up in the page. Currently: convert NameObject nameddestination to TextStringObject (required for names/dests list)

   Parameters page –

   Returns The cleaned PageObject

append(*fileobj: Union[str, IO[Any], PdfReader, Path]*, *outline_item: Union[str, None, PageRange, Tuple[int, int], Tuple[int, int, int], List[int]] = None*, *pages: Union[None, PageRange, Tuple[int, int], Tuple[int, int, int], List[int], List[PageObject]] = None*, *import_outline: bool = True*, *excluded_fields: Optional[Union[List[str], Tuple[str, ...]]] = None*) → None

Identical to the *merge()* method, but assumes you want to concatenate all pages onto the end of the file instead of specifying a position.

Parameters

- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.

- **outline_item** – Optionally, you may specify a string to build an outline (aka 'bookmark') to identify the beginning of the included file.

- **pages** – Can be a *PageRange* or a (start, stop[, step]) tuple or a list of pages to be processed to merge only the specified range of pages from the source document into the output document.

- **import_outline** – You may prevent the source document's outline (collection of outline items, previously referred to as 'bookmarks') from being imported by specifying this as False.

- **excluded_fields** – Provide the list of fields/keys to be ignored if /Annots is part of the list, the annotation will be ignored if /B is part of the list, the articles will be ignored

merge(*position: Optional[int]*, *fileobj: Union[Path, str, IO[Any], PdfReader]*, *outline_item: Optional[str] = None*, *pages: Optional[Union[str, PageRange, Tuple[int, int], Tuple[int, int, int], List[int], List[PageObject]]] = None*, *import_outline: bool = True*, *excluded_fields: Optional[Union[List[str], Tuple[str, ...]]] = ()*) → None

Merge the pages from the given file into the output file at the specified page number.

Parameters

- **position** – The *page number* to insert this file. File will be inserted after the given number.

- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.

- **outline_item** – Optionally, you may specify a string to build an outline (aka 'bookmark') to identify the beginning of the included file.

- **pages** – can be a *PageRange* or a (start, stop[, step]) tuple or a list of pages to be processed to merge only the specified range of pages from the source document into the output document.

- **import_outline** – You may prevent the source document's outline (collection of outline items, previously referred to as 'bookmarks') from being imported by specifying this as False.

- **excluded_fields** – provide the list of fields/keys to be ignored if /Annots is part of the list, the annotation will be ignored if /B is part of the list, the articles will be ignored

Raises **TypeError** – The pages attribute is not configured properly

add_filtered_articles(*fltr: Union[Pattern, str]*, *pages: Dict[int, PageObject]*, *reader: PdfReader*) → None

Add articles matching the defined criteria.

Parameters

- **fltr** –

- **pages** –

- **reader** –

close() → None

> To match the functions from Merger.

find_outline_item(*outline_item:* *Dict[str, Any]*, *root:* *Optional[List[Union[*Destination,
*List[Union[*Destination, *List[*Destination*]]]]]* = *None*) → Optional[List[int]]

find_bookmark(*outline_item:* *Dict[str, Any]*, *root:* *Optional[List[Union[*Destination,
*List[Union[*Destination, *List[*Destination*]]]]]* = *None*) → Optional[List[int]]

> Deprecated since version 2.9.0: Use `find_outline_item()` instead.

reset_translation(*reader:* *Union[None,* PdfReader, *IndirectObject]* = *None*) → None

> Reset the translation table between reader and the writer object.
>
> Late cloning will create new independent objects.
>
> > **Parameters** `reader` – PdfReader or IndirectObject refering a PdfReader object. if set to None
> > or omitted, all tables will be reset.

set_page_label(*page_index_from:* *int*, *page_index_to:* *int*, *style:* *Optional[PageLabelStyle]* = *None*, *prefix:*
Optional[str] = *None*, *start:* *Optional[int]* = *0*) → None

> Set a page label to a range of pages.
>
> Page indexes must be given starting from 0. Labels must have a style, a prefix or both. If to a range is not
> assigned any page label a decimal label starting from 1 is applied.
>
> > **Parameters**
> >
> > - `page_index_from` – page index of the beginning of the range starting from 0
> >
> > - `page_index_to` – page index of the beginning of the range starting from 0
> >
> > - `style` – The numbering style to be used for the numeric portion of each page label: '/D'
> >   Decimal arabic numerals '/R' Uppercase roman numerals '/r' Lowercase roman numerals
> >   '/A' Uppercase letters (A to Z for the first 26 pages,
> >
> > > AA to ZZ for the next 26, and so on)
> >
> > **'/a' Lowercase letters (a to z for the first 26 pages,** aa to zz for the next 26, and so on)
> >
> > - `prefix` – The label prefix for page labels in this range.
> >
> > - `start` – The value of the numeric portion for the first page label in the range. Subsequent
> >   pages are numbered sequentially from this value, which must be greater than or equal to 1.
> >   Default value: 1.

THE PDFMERGER CLASS

class pypdf.**PdfMerger**(*strict:* [*bool*](#) *= False*, *fileobj:* [*Union[Path, str, IO[Any]]*](#) *= '')*

> Bases: [object](#)
>
> Use [*PdfWriter*](#) instead.
>
> Deprecated since version 5.0.0.
>
> **merge**(*page_number:* [*Optional[int]*](#) *= None*, *fileobj:* [*Union[None, Path, str, IO[Any]*](#), [PdfReader](#)*] = None*,
>   *outline_item:* [*Optional[str]*](#) *= None*, *pages:* [*Optional[Union[str*](#), [PageRange](#), *Tuple[int, int]*, *Tuple[int*,
>   *int, int]*, *List[int]]]*](#) *= None*, *import_outline:* [*bool*](#) *= True*, *position:* [*Optional[int]*](#) *= None*) → [None](#)
>
> > Merge the pages from the given file into the output file at the specified page number.
> >
> > > **Parameters**
> > >
> > > - **page_number** – The *page number* to insert this file. File will be inserted after the given number.
> > >
> > > - **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file. None as an argument is deprecated.
> > >
> > > - **outline_item** – Optionally, you may specify an outline item (previously referred to as a 'bookmark') to be applied at the beginning of the included file by supplying the text of the outline item.
> > >
> > > - **pages** –
> > >
> > >   **can be a** [*PageRange*](#) or a (start, stop[, step]) tuple to merge only the specified range of pages from the source document into the output document. Can also be a list of pages to merge.
> > >
> > >   **import_outline: You may prevent the source document's** outline (collection of outline items, previously referred to as 'bookmarks') from being imported by specifying this as False.
>
> **append**(*fileobj:* [*Union[str*](#), [*IO[Any]*](#), [PdfReader](#), [*Path*](#)*]*, *outline_item:* [*Optional[str]*](#) *= None*, *pages:*
>   [*Union[None*](#), [PageRange](#), *Tuple[int, int]*, *Tuple[int, int, int]*, *List[int]]*](#) *= None*, *import_outline:* [*bool*](#) *=*
>   *True*) → [None](#)
>
> > Identical to the [merge()](#) method, but assumes you want to concatenate all pages onto the end of the file instead of specifying a position.
> >
> > > **Parameters**
> > >
> > > - **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.

- **outline_item** – Optionally, you may specify an outline item (previously referred to as a 'bookmark') to be applied at the beginning of the included file by supplying the text of the outline item.

- **pages** – can be a *PageRange* or a (`start, stop[, step]`) tuple to merge only the specified range of pages from the source document into the output document. Can also be a list of pages to append.

- **import_outline** – You may prevent the source document's outline (collection of outline items, previously referred to as 'bookmarks') from being imported by specifying this as `False`.

write(*fileobj: Union[Path, str, IO[Any]]*) → None

Write all data that has been merged to the given output file.

> **Parameters fileobj** – Output file. Can be a filename or any kind of file-like object.

close() → None

Shut all file descriptors (input and output) and clear all memory usage.

add_metadata(*infos: Dict[str, Any]*) → None

Add custom metadata to the output.

> **Parameters infos** – a Python dictionary where each key is a field and each value is your new metadata. An example is {'/Title': 'My title'}

addMetadata(*infos: Dict[str, Any]*) → None

Use *add_metadata()* instead.

Deprecated since version 1.28.0.

setPageLayout(*layout: Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']*) → None

Use *set_page_layout()* instead.

Deprecated since version 1.28.0.

set_page_layout(*layout: Literal['/NoLayout', '/SinglePage', '/OneColumn', '/TwoColumnLeft', '/TwoColumnRight', '/TwoPageLeft', '/TwoPageRight']*) → None

Set the page layout.

> **Parameters layout** – The page layout to be used

Table 1: Valid `layout` arguments

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

setPageMode(*mode: Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']*) → None

Use *set_page_mode()* instead.

Deprecated since version 1.28.0.

set_page_mode(*mode: Literal['/UseNone', '/UseOutlines', '/UseThumbs', '/FullScreen', '/UseOC', '/UseAttachments']*) → None

>   Set the page mode.

>>   **Parameters mode** – The page mode to use.

Table 2: Valid mode arguments

/UseNone	Do not show outline or thumbnails panels
/UseOutlines	Show outline (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

find_outline_item(*outline_item: Dict[str, Any]*, *root: Optional[List[Union[Destination, List[Union[Destination, List[Destination]]]]]] = None*) → Optional[List[int]]

find_bookmark(*outline_item: Dict[str, Any]*, *root: Optional[List[Union[Destination, List[Union[Destination, List[Destination]]]]]] = None*) → Optional[List[int]]

>   Deprecated since version 2.9.0: Use `find_outline_item()` instead.

add_outline_item(*title: str*, *page_number: ~typing.Optional[int] = None*, *parent: ~typing.Union[None, ~pypdf.generic._data_structures.TreeObject, ~pypdf.generic._base.IndirectObject] = None*, *color: ~typing.Optional[~typing.Tuple[float, float, float]] = None*, *bold: bool = False*, *italic: bool = False*, *fit: ~pypdf.generic._fit.Fit = <pypdf.generic._fit.Fit object>*, *pagenum: ~typing.Optional[int] = None*) → IndirectObject

>   Add an outline item (commonly referred to as a "Bookmark") to this PDF file.

>>   **Parameters**

>>   - **title** – Title to use for this outline item.

>>   - **page_number** – Page number this outline item will point to.

>>   - **parent** – A reference to a parent outline item to create nested outline items.

>>   - **color** – Color of the outline item's font as a red, green, blue tuple from 0.0 to 1.0

>>   - **bold** – Outline item font is bold

>>   - **italic** – Outline item font is italic

>>   - **fit** – The fit of the destination page.

addBookmark(*title: str*, *pagenum: int*, *parent: Union[None, TreeObject, IndirectObject] = None*, *color: Optional[Tuple[float, float, float]] = None*, *bold: bool = False*, *italic: bool = False*, *fit: Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit'*, **args: Union[NumberObject, NullObject, float]*) → IndirectObject

>   Deprecated since version 1.28.0: Use `add_outline_item()` instead.

add_bookmark(*title: str*, *pagenum: int*, *parent: Union[None, TreeObject, IndirectObject] = None*, *color: Optional[Tuple[float, float, float]] = None*, *bold: bool = False*, *italic: bool = False*, *fit: Literal['/Fit', '/XYZ', '/FitH', '/FitV', '/FitR', '/FitB', '/FitBH', '/FitBV'] = '/Fit'*, **args: Union[NumberObject, NullObject, float]*) → IndirectObject

>   Deprecated since version 2.9.0: Use `add_outline_item()` instead.

addNamedDestination(*title: str*, *pagenum: int*) → None

>   Deprecated since version 1.28.0: Use `add_named_destination()` instead.

add_named_destination(*title:* *str*, *page_number:* *Optional*[*int*] *= None*, *pagenum:* *Optional*[*int*] *= None*)
→ None

Add a destination to the output.

Parameters

- **title** – Title to use

- **page_number** – Page number this destination points at.

THE PAGEOBJECT CLASS

class pypdf._page.**PageObject**(*pdf: Union[None, PdfReaderProtocol, PdfWriterProtocol] = None*, *indirect_reference: Optional[IndirectObject] = None*, *indirect_ref: Optional[IndirectObject] = None*)

Bases: `DictionaryObject`

PageObject represents a single page within a PDF file.

Typically this object will be created by accessing the `get_page()` method of the *PdfReader* class, but it is also possible to create an empty page with the *create_blank_page()* static method.

> **Parameters**
>
> - **pdf** – PDF file the page belongs to.
> - **indirect_reference** – Stores the original indirect reference to this object in its source PDF

original_page: *PageObject*

property indirect_ref: Optional[pypdf.generic._base.IndirectObject]

hash_value_data() → bytes

property user_unit: float

> A read-only positive number giving the size of user space units.
>
> It is in multiples of 1/72 inch. Hence a value of 1 means a user space unit is 1/72 inch, and a value of 3 means that a user space unit is 3/72 inch.

static create_blank_page(*pdf: Union[None, PdfReaderProtocol, PdfWriterProtocol] = None*, *width: Optional[Union[float, Decimal]] = None*, *height: Optional[Union[float, Decimal]] = None*) → *PageObject*

> Return a new blank page.
>
> If `width` or `height` is None, try to get the page size from the last page of *pdf*.
>
> > **Parameters**
> >
> > - **pdf** – PDF file the page belongs to
> > - **width** – The width of the new page expressed in default user space units.
> > - **height** – The height of the new page expressed in default user space units.
>
> **Returns** The new blank page
>
> **Raises** **PageSizeNotDefinedError** – if pdf is None or contains no page

static createBlankPage(*pdf:* *Optional[PdfReaderProtocol] = None*, *width:* *Optional[Union[float,* *Decimal]] = None*, *height:* *Optional[Union[float, Decimal]] = None*) → *PageObject*

> Use `create_blank_page()` instead.

> Deprecated since version 1.28.0.

property images: List[pypdf._utils.ImageFile]

> Read-only property emulating a list of images on a page.

> Get a list of all images on the page. The key can be: - A string (for the top object) - A tuple (for images within XObject forms) - An integer

Examples

> reader.pages[0].images[0] # return fist image reader.pages[0].images['/I0'] # return image '/I0' # return image '/Image1' within '/TP1' Xobject/Form: reader.pages[0].images['/TP1','/Image1'] for img in reader.pages[0].images: # loop within all objects

> images.keys() and images.items() can be used.

> The ImageFile has the following properties:

> > *.name* : name of the object *.data* : bytes of the object *.image* : PIL Image Object *.indirect_reference* : object reference

> **and the following methods:**

> > *.replace(new_image: PIL.Image.Image, **kwargs)* : replace the image in the pdf with the new image applying the saving parameters indicated (such as quality)

> Example usage:

> > reader.pages[0].images[0]=replace(Image.open("new_image.jpg", quality = 20)

> Inline images are extracted and named ~0~, ~1~, ..., with the indirect_reference set to None.

property rotation: int

> The VISUAL rotation of the page.

> This number has to be a multiple of 90 degrees: 0, 90, 180, or 270 are valid values. This property does not affect /Contents.

transfer_rotation_to_content() → None

> Apply the rotation of the page to the content and the media/crop/... boxes.

> It's recommended to apply this function before page merging.

rotate(*angle:* *int*) → *PageObject*

> Rotate a page clockwise by increments of 90 degrees.

> > **Parameters** `angle` – Angle to rotate the page. Must be an increment of 90 deg.

> > **Returns** The rotated PageObject

rotate_clockwise(*angle:* *int*) → *PageObject*

rotateClockwise(*angle:* *int*) → *PageObject*

> Use `rotate_clockwise()` instead.

> Deprecated since version 1.28.0.

rotateCounterClockwise(*angle:* [*int*](#)) → [*PageObject*](#)

> Use [`rotate_clockwise()`](#) with a negative argument instead.
>
> Deprecated since version 1.28.0.

get_contents() → [Optional](#)[ContentStream]

> Access the page contents.
>
> > **Returns** The `/Contents` object, or `None` if it doesn't exist. `/Contents` is optional, as described
> > in PDF Reference 7.7.3.3

getContents() → [Optional](#)[ContentStream]

> Use [`get_contents()`](#) instead.
>
> Deprecated since version 1.28.0.

replace_contents(*content:* [*Union[None, ContentStream, EncodedStreamObject, ArrayObject]*](#)) → [None](#)

> Replace the page contents with the new content and nullify old objects :param content: new content. if
> None delete the content field.

merge_page(*page2:* [PageObject](#), *expand:* [*bool*](#) *= False*, *over:* [*bool*](#) *= True*) → [None](#)

> Merge the content streams of two pages into one.
>
> Resource references (i.e. fonts) are maintained from both pages. The mediabox/cropbox/etc of this page
> are not altered. The parameter page's content stream will be added to the end of this page's content stream,
> meaning that it will be drawn after, or "on top" of this page.
>
> > **Parameters**
> >
> > - **page2** – The page to be merged into this one. Should be an instance of [`PageObject`](#).
> >
> > - **over** – set the page2 content over page1 if True(default) else under
> >
> > - **expand** – If true, the current page dimensions will be expanded to accommodate the dimensions of the page to be merged.

mergePage(*page2:* [PageObject](#)) → [None](#)

> Use [`merge_page()`](#) instead.
>
> Deprecated since version 1.28.0.

merge_transformed_page(*page2:* [PageObject](#), *ctm:* [*Union[Tuple[float, float, float, float, float, float],*](#) [*Transformation]*](#), *over:* [*bool*](#) *= True*, *expand:* [*bool*](#) *= False*) → [None](#)

> merge_transformed_page is similar to merge_page, but a transformation matrix is applied to the merged
> stream.
>
> > **Parameters**
> >
> > - **page2** – The page to be merged into this one.
> >
> > - **ctm** – a 6-element tuple containing the operands of the transformation matrix
> >
> > - **over** – set the page2 content over page1 if True(default) else under
> >
> > - **expand** – Whether the page should be expanded to fit the dimensions of the page to be merged.

mergeTransformedPage(*page2:* [PageObject](#), *ctm:* [*Union[Tuple[float, float, float, float, float, float],*](#) [*Transformation]*](#), *expand:* [*bool*](#) *= False*) → [None](#)

> deprecated
>
> deprecated:: 1.28.0

Use `merge_transformed_page()` instead.

merge_scaled_page(*page2:* PageObject, *scale: float*, *over: bool = True*, *expand: bool = False*) → None

merge_scaled_page is similar to merge_page, but the stream to be merged is scaled by applying a transformation matrix.

> **Parameters**
>
> - **page2** – The page to be merged into this one.
>
> - **scale** – The scaling factor
>
> - **over** – set the page2 content over page1 if True(default) else under
>
> - **expand** – Whether the page should be expanded to fit the dimensions of the page to be merged.

mergeScaledPage(*page2:* PageObject, *scale: float*, *expand: bool = False*) → None

deprecated

Deprecated since version 1.28.0: Use `merge_scaled_page()` instead.

merge_rotated_page(*page2:* PageObject, *rotation: float*, *over: bool = True*, *expand: bool = False*) → None

merge_rotated_page is similar to merge_page, but the stream to be merged is rotated by applying a transformation matrix.

> **Parameters**
>
> - **page2** – The page to be merged into this one.
>
> - **rotation** – The angle of the rotation, in degrees
>
> - **over** – set the page2 content over page1 if True(default) else under
>
> - **expand** – Whether the page should be expanded to fit the dimensions of the page to be merged.

mergeRotatedPage(*page2:* PageObject, *rotation: float*, *expand: bool = False*) → None

deprecated

Deprecated since version 1.28.0: Use `add_transformation()` and `merge_page()` instead.

merge_translated_page(*page2:* PageObject, *tx: float*, *ty: float*, *over: bool = True*, *expand: bool = False*)
    → None

mergeTranslatedPage is similar to merge_page, but the stream to be merged is translated by applying a transformation matrix.

> **Parameters**
>
> - **page2** – the page to be merged into this one.
>
> - **tx** – The translation on X axis
>
> - **ty** – The translation on Y axis
>
> - **over** – set the page2 content over page1 if True(default) else under
>
> - **expand** – Whether the page should be expanded to fit the dimensions of the page to be merged.

mergeTranslatedPage(*page2:* PageObject, *tx: float*, *ty: float*, *expand: bool = False*) → None

deprecated

Deprecated since version 1.28.0: Use `merge_translated_page()` instead.

mergeRotatedTranslatedPage(*page2:* PageObject, *rotation:* *float*, *tx:* *float*, *ty:* *float*, *expand:* *bool* = *False*) → None

>Deprecated since version 1.28.0: Use `merge_transformed_page()` instead.

mergeRotatedScaledPage(*page2:* PageObject, *rotation:* *float*, *scale:* *float*, *expand:* *bool* = *False*) → None

>Deprecated since version 1.28.0: Use `merge_transformed_page()` instead.

mergeScaledTranslatedPage(*page2:* PageObject, *scale:* *float*, *tx:* *float*, *ty:* *float*, *expand:* *bool* = *False*) → None

>mergeScaledTranslatedPage is similar to merge_page, but the stream to be merged is translated and scaled by applying a transformation matrix.
>
>>**Parameters**
>>
>>- **page2** (PageObject) – the page to be merged into this one. Should be an instance of PageObject.
>>- **scale** (float) – The scaling factor
>>- **tx** (float) – The translation on X axis
>>- **ty** (float) – The translation on Y axis
>>- **expand** (bool) – Whether the page should be expanded to fit the dimensions of the page to be merged.
>
>Deprecated since version 1.28.0: Use `add_transformation()` and `merge_page()` instead.

mergeRotatedScaledTranslatedPage(*page2:* PageObject, *rotation:* *float*, *scale:* *float*, *tx:* *float*, *ty:* *float*, *expand:* *bool* = *False*) → None

>mergeRotatedScaledTranslatedPage is similar to merge_page, but the stream to be merged is translated, rotated and scaled by applying a transformation matrix.
>
>>**Parameters**
>>
>>- **page2** (PageObject) – the page to be merged into this one. Should be an instance of PageObject.
>>- **tx** (float) – The translation on X axis
>>- **ty** (float) – The translation on Y axis
>>- **rotation** (float) – The angle of the rotation, in degrees
>>- **scale** (float) – The scaling factor
>>- **expand** (bool) – Whether the page should be expanded to fit the dimensions of the page to be merged.
>
>Deprecated since version 1.28.0: Use `add_transformation()` and `merge_page()` instead.

add_transformation(*ctm:* Union[Transformation, Tuple[float, float, float, float, float, float]], *expand:* *bool* = *False*) → None

>Apply a transformation matrix to the page.
>
>>**Parameters** **ctm** – A 6-element tuple containing the operands of the transformation matrix. Alternatively, a `Transformation` object can be passed.
>
>See *Cropping and Transforming PDFs*.

addTransformation(*ctm:* Tuple[float, float, float, float, float, float]) → None

>Use `add_transformation()` instead.
>
>Deprecated since version 1.28.0.

scale(*sx: float*, *sy: float*) → None

Scale a page by the given factors by applying a transformation matrix to its content and updating the page size.

This updates the mediabox, the cropbox, and the contents of the page.

> **Parameters**
>
> - **sx** – The scaling factor on horizontal axis.
>
> - **sy** – The scaling factor on vertical axis.

scale_by(*factor: float*) → None

Scale a page by the given factor by applying a transformation matrix to its content and updating the page size.

> **Parameters factor** – The scaling factor (for both X and Y axis).

scaleBy(*factor: float*) → None

Use *scale_by()* instead.

Deprecated since version 1.28.0.

scale_to(*width: float*, *height: float*) → None

Scale a page to the specified dimensions by applying a transformation matrix to its content and updating the page size.

> **Parameters**
>
> - **width** – The new width.
>
> - **height** – The new height.

scaleTo(*width: float*, *height: float*) → None

Use *scale_to()* instead.

Deprecated since version 1.28.0.

compress_content_streams(*level: int = - 1*) → None

Compress the size of this page by joining all content streams and applying a FlateDecode filter.

However, it is possible that this function will perform no action if content stream compression becomes "automatic".

compressContentStreams() → None

Use *compress_content_streams()* instead.

Deprecated since version 1.28.0.

property page_number: int

Read-only property which return the page number with the pdf file.

> **Returns** *int* – page number ; -1 if the page is not attached to a pdf

extract_text(**args: Any*, *Tj_sep: Optional[str] = None*, *TJ_sep: Optional[str] = None*, *orientations: Union[int, Tuple[int, ...]] = (0, 90, 180, 270)*, *space_width: float = 200.0*, *visitor_operand_before: Optional[Callable[[Any, Any, Any, Any], None]] = None*, *visitor_operand_after: Optional[Callable[[Any, Any, Any, Any], None]] = None*, *visitor_text: Optional[Callable[[Any, Any, Any, Any, Any], None]] = None*) → str

Locate all text drawing commands, in the order they are provided in the content stream, and extract the text.

This works well for some PDF files, but poorly for others, depending on the generator used. This will be refined in the future.

Do not rely on the order of text coming out of this function, as it will change if this function is made more sophisticated.

Arabic, Hebrew,… are extracted in the good order. If required an custom RTL range of characters can be defined; see function set_custom_rtl

Additionally you can provide visitor-methods to get informed on all operations and all text-objects. For example in some PDF files this can be useful to parse tables.

> **Parameters**
>
> - **Tj_sep** – Deprecated. Kept for compatibility until pypdf 4.0.0
>
> - **TJ_sep** – Deprecated. Kept for compatibility until pypdf 4.0.0
>
> - **orientations** – list of orientations text_extraction will look for default = (0, 90, 180, 270) note: currently only 0(Up),90(turned Left), 180(upside Down), 270 (turned Right)
>
> - **space_width** – force default space width if not extracted from font (default: 200)
>
> - **visitor_operand_before** – function to be called before processing an operation. It has four arguments: operator, operand-arguments, current transformation matrix and text matrix.
>
> - **visitor_operand_after** – function to be called after processing an operation. It has four arguments: operator, operand-arguments, current transformation matrix and text matrix.
>
> - **visitor_text** – function to be called when extracting some text at some position. It has five arguments: text, current transformation matrix, text matrix, font-dictionary and font-size. The font-dictionary may be None in case of unknown fonts. If not None it may e.g. contain key "/BaseFont" with value "/Arial,Bold".
>
> **Returns** The extracted text

extract_xform_text(*xform: EncodedStreamObject*, *orientations:* Tuple*[*int*, ...] = (0, 90, 270, 360)*, *space_width:* float *= 200.0*, *visitor_operand_before:* Optional*[*Callable*[[*Any*,* Any*,* Any*,* Any*]*, None*]] = None*, *visitor_operand_after:* Optional*[*Callable*[[*Any*,* Any*,* Any*,* Any*]*, None*]] = None*, *visitor_text:* Optional*[*Callable*[[*Any*,* Any*,* Any*,* Any*,* Any*]*, None*]] = None*) → str

Extract text from an XObject.

> **Parameters**
>
> - **xform** –
>
> - **orientations** –
>
> - **space_width** – force default space width (if not extracted from font (default 200)
>
> - **visitor_operand_before** –
>
> - **visitor_operand_after** –
>
> - **visitor_text** –
>
> **Returns** The extracted text

extractText(*Tj_sep:* str *= ''*, *TJ_sep:* str *= ''*) → str

Use extract_text() instead.

Deprecated since version 1.28.0.

property mediabox

A *RectangleObject*, expressed in default user space units, defining the boundaries of the physical medium on which the page is intended to be displayed or printed.

property mediaBox: *RectangleObject*

Use *mediabox* instead.

Deprecated since version 1.28.0.

property cropbox

A *RectangleObject*, expressed in default user space units, defining the visible region of default user space.

When the page is displayed or printed, its contents are to be clipped (cropped) to this rectangle and then imposed on the output medium in some implementation-defined manner. Default value: same as *mediabox*.

property cropBox: *RectangleObject*

Use *cropbox* instead.

Deprecated since version 1.28.0.

property bleedbox

A *RectangleObject*, expressed in default user space units, defining the region to which the contents of the page should be clipped when output in a production environment.

property bleedBox: *RectangleObject*

Use *bleedbox* instead.

Deprecated since version 1.28.0.

property trimbox

A *RectangleObject*, expressed in default user space units, defining the intended dimensions of the finished page after trimming.

property trimBox: *RectangleObject*

Use *trimbox* instead.

Deprecated since version 1.28.0.

property artbox

A *RectangleObject*, expressed in default user space units, defining the extent of the page's meaningful content as intended by the page's creator.

property artBox: *RectangleObject*

Use *artbox* instead.

Deprecated since version 1.28.0.

property annotations: Optional[pypdf.generic._data_structures.ArrayObject]

THE TRANSFORMATION CLASS

class pypdf.**Transformation**(*ctm: Tuple[float, float, float, float, float, float] = (1, 0, 0, 1, 0, 0)*)

> Bases: object

> Represent a 2D transformation.

> The transformation between two coordinate systems is represented by a 3-by-3 transformation matrix matrix with the following form:

```
a b 0
c d 0
e f 1
```

> Because a transformation matrix has only six elements that can be changed, it is usually specified in PDF as the six-element array [a b c d e f].

> Coordinate transformations are expressed as matrix multiplications:

```
                    a b 0
[ x y 1 ] = [ x y 1 ] ×   c d 0
                    e f 1
```

> **Example**

```
>>> from pypdf import Transformation
>>> op = Transformation().scale(sx=2, sy=3).translate(tx=10, ty=20)
>>> page.add_transformation(op)
```

> **property matrix:** Tuple[Tuple[float, float, float], Tuple[float, float, float], Tuple[float, float, float]]

>> Return the transformation matrix as a tuple of tuples in the form:

>> ((a, b, 0), (c, d, 0), (e, f, 1))

> **static compress**(*matrix: Tuple[Tuple[float, float, float], Tuple[float, float, float], Tuple[float, float, float]]*) → Tuple[float, float, float, float, float, float]

> Compresses the transformation matrix into a tuple of (a, b, c, d, e, f).

>> **Parameters matrix** – The transformation matrix as a tuple of tuples.

>> **Returns** A tuple representing the transformation matrix as (a, b, c, d, e, f)

transform(*m:* Transformation) → *Transformation*

> Apply one transformation to another.
>
> > **Parameters** `m` – a Transformation to apply.
> >
> > **Returns** A new `Transformation` instance

Example

```
>>> from pypdf import Transformation
>>> op = Transformation((1, 0, 0, -1, 0, height)) # vertical mirror
>>> op = Transformation().transform(Transformation((-1, 0, 0, 1, iwidth, 0))) #
→horizontal mirror
>>> page.add_transformation(op)
```

translate(*tx: float = 0, ty: float = 0*) → *Transformation*

> Translate the contents of a page.
>
> > **Parameters**
> >
> > - `tx` – The translation along the x-axis.
> >
> > - `ty` – The translation along the y-axis.
> >
> > **Returns** A new `Transformation` instance

scale(*sx: Optional[float] = None, sy: Optional[float] = None*) → *Transformation*

> Scale the contents of a page towards the origin of the coordinate system.
>
> Typically, that is the lower-left corner of the page. That can be changed by translating the contents / the page boxes.
>
> > **Parameters**
> >
> > - `sx` – The scale factor along the x-axis.
> >
> > - `sy` – The scale factor along the y-axis.
> >
> > **Returns** A new Transformation instance with the scaled matrix.

rotate(*rotation: float*) → *Transformation*

> Rotate the contents of a page.
>
> > **Parameters** `rotation` – The angle of rotation in degrees.
> >
> > **Returns** A new `Transformation` instance with the rotated matrix.

apply_on(*pt: Union[Tuple[float, float], List[float]], as_object: bool = False*) → Union[Tuple[float, float], List[float]]

> Apply the transformation matrix on the given point.
>
> > **Parameters** `pt` – A tuple or list representing the point in the form (x, y)
> >
> > **Returns** A tuple or list representing the transformed point in the form (x', y')

THE DOCUMENTINFORMATION CLASS

class pypdf.**DocumentInformation**

Bases: `DictionaryObject`

A class representing the basic document metadata provided in a PDF File. This class is accessible through *PdfReader.metadata*.

All text properties of the document metadata have *two* properties, eg. author and author_raw. The non-raw property will always return a `TextStringObject`, making it ideal for a case where the metadata is being displayed. The raw property can sometimes return a `ByteStringObject`, if pypdf was unable to decode the string's text encoding; this requires additional safety in the caller and therefore is not as commonly accessed.

getText(*key: str*) → Optional[str]

Use the attributes (e.g. *title* / *author*).

Deprecated since version 1.28.0.

property title: Optional[str]

Read-only property accessing the document's title.

Returns a `TextStringObject` or `None` if the title is not specified.

property title_raw: Optional[str]

The "raw" version of title; can return a `ByteStringObject`.

property author: Optional[str]

Read-only property accessing the document's author.

Returns a `TextStringObject` or `None` if the author is not specified.

property author_raw: Optional[str]

The "raw" version of author; can return a `ByteStringObject`.

property subject: Optional[str]

Read-only property accessing the document's subject.

Returns a `TextStringObject` or `None` if the subject is not specified.

property subject_raw: Optional[str]

The "raw" version of subject; can return a `ByteStringObject`.

property creator: Optional[str]

Read-only property accessing the document's creator.

If the document was converted to PDF from another format, this is the name of the application (e.g. OpenOffice) that created the original document from which it was converted. Returns a `TextStringObject` or `None` if the creator is not specified.

property creator_raw: Optional[str]

    The "raw" version of creator; can return a `ByteStringObject`.

property producer: Optional[str]

    Read-only property accessing the document's producer.

    If the document was converted to PDF from another format, this is the name of the application (for example, OSX Quartz) that converted it to PDF. Returns a `TextStringObject` or `None` if the producer is not specified.

property producer_raw: Optional[str]

    The "raw" version of producer; can return a `ByteStringObject`.

property creation_date: Optional[datetime]

    Read-only property accessing the document's creation date.

property creation_date_raw: Optional[str]

    The "raw" version of creation date; can return a `ByteStringObject`.

    Typically in the format `D:YYYYMMDDhhmmss[+Z-]hh'mm` where the suffix is the offset from UTC.

property modification_date: Optional[datetime]

    Read-only property accessing the document's modification date.

    The date and time the document was most recently modified.

property modification_date_raw: Optional[str]

    The "raw" version of modification date; can return a `ByteStringObject`.

    Typically in the format `D:YYYYMMDDhhmmss[+Z-]hh'mm` where the suffix is the offset from UTC.

THE XMPINFORMATION CLASS

class pypdf.xmp.**XmpInformation**(*stream: ContentStream*)

Bases: `PdfObject`

An object that represents Adobe XMP metadata. Usually accessed by *xmp_metadata()*

> **Raises** `PdfReadError` – if XML is invalid

property rdfRoot: `xml.dom.minidom.Element`

write_to_stream(*stream: IO[Any]*, *encryption_key: Union[None, str, bytes] = None*) → None

writeToStream(*stream: IO[Any]*, *encryption_key: Union[None, str, bytes]*) → None

> Use *write_to_stream()* instead.
>
> Deprecated since version 1.28.0.

get_element(*about_uri: str*, *namespace: str*, *name: str*) → Iterator[Any]

getElement(*aboutUri: str*, *namespace: str*, *name: str*) → Iterator[Any]

> Use *get_element()* instead.
>
> Deprecated since version 1.28.0.

get_nodes_in_namespace(*about_uri: str*, *namespace: str*) → Iterator[Any]

getNodesInNamespace(*aboutUri: str*, *namespace: str*) → Iterator[Any]

> Use *get_nodes_in_namespace()* instead.
>
> Deprecated since version 1.28.0.

property dc_contributor: `Optional[List[str]]`

> Contributors to the resource (other than the authors).
>
> An unsorted array of names.

property dc_coverage: `Optional[Any]`

> Text describing the extent or scope of the resource.

property dc_creator: `Optional[List[Any]]`

> A sorted array of names of the authors of the resource, listed in order of precedence.

property dc_date: `Optional[List[Any]]`

> A sorted array of dates (datetime.datetime instances) of significance to the resource.
>
> The dates and times are in UTC.

property dc_description: Optional[Dict[Any, Any]]

    A language-keyed dictionary of textual descriptions of the content of the resource.

property dc_format: Optional[Any]

    The mime-type of the resource.

property dc_identifier: Optional[Any]

    Unique identifier of the resource.

property dc_language: Optional[List[str]]

    An unordered array specifying the languages used in the resource.

property dc_publisher: Optional[List[str]]

    An unordered array of publisher names.

property dc_relation: Optional[List[str]]

    An unordered array of text descriptions of relationships to other documents.

property dc_rights: Optional[Dict[Any, Any]]

    A language-keyed dictionary of textual descriptions of the rights the user has to this resource.

property dc_source: Optional[Any]

    Unique identifier of the work from which this resource was derived.

property dc_subject: Optional[List[str]]

    An unordered array of descriptive phrases or keywrods that specify the topic of the content of the resource.

property dc_title: Optional[Dict[Any, Any]]

    A language-keyed dictionary of the title of the resource.

property dc_type: Optional[List[str]]

    An unordered array of textual descriptions of the document type.

property pdf_keywords: Optional[Any]

    An unformatted text string representing document keywords.

property pdf_pdfversion: Optional[Any]

    The PDF file version, for example 1.0, 1.3.

property pdf_producer: Optional[Any]

    The name of the tool that created the PDF document.

property xmp_create_date: Optional[Any]

    The date and time the resource was originally created.

    The date and time are returned as a UTC datetime.datetime object.

property xmp_createDate: datetime

property xmp_modify_date: Optional[Any]

    The date and time the resource was last modified.

    The date and time are returned as a UTC datetime.datetime object.

property xmp_modifyDate: datetime

property xmp_metadata_date: Optional[Any]

    The date and time that any metadata for this resource was last changed.

    The date and time are returned as a UTC datetime.datetime object.

property xmp_metadataDate: `datetime`

property xmp_creator_tool: `Optional[Any]`

> The name of the first known tool used to create the resource.

property xmp_creatorTool: `str`

property xmpmm_document_id: `Optional[Any]`

> The common identifier for all versions and renditions of this resource.

property xmpmm_documentId: `str`

property xmpmm_instance_id: `Optional[Any]`

> An identifier for a specific incarnation of a document, updated each time a file is saved.

property xmpmm_instanceId: `str`

property custom_properties: `Dict[Any, Any]`

> Retrieve custom metadata properties defined in the undocumented pdfx metadata schema.
>
> > **Returns** A dictionary of key/value items for custom metadata properties.

THE DESTINATION CLASS

class pypdf.generic.**Destination**(*title: str*, *page: Union[NumberObject, IndirectObject, NullObject, DictionaryObject]*, *fit:* Fit)

Bases: `TreeObject`

A class representing a destination within a PDF file.

See section 8.2.1 of the PDF 1.6 reference.

> **Parameters**
>> • **title** – Title of this destination.
>>
>> • **page** – Reference to the page of this destination. Should be an instance of `IndirectObject`.
>>
>> • **fit** – How the destination is displayed.
>
> **Raises** `PdfReadError` – If destination type is invalid.

node: Optional[pypdf.generic._data_structures.DictionaryObject] = None

childs: List[Any] = []

property dest_array: pypdf.generic._data_structures.ArrayObject

getDestArray() → ArrayObject

> Use *dest_array* instead.

> Deprecated since version 1.28.3.

write_to_stream(*stream: IO[Any]*, *encryption_key: Union[None, str, bytes] = None*) → None

property title: Optional[str]

> Read-only property accessing the destination title.

property page: Optional[int]

> Read-only property accessing the destination page number.

property typ: Optional[str]

> Read-only property accessing the destination type.

property zoom: Optional[int]

> Read-only property accessing the zoom factor.

property left: Optional[pypdf.generic._base.FloatObject]

> Read-only property accessing the left horizontal coordinate.

property right: Optional[pypdf.generic._base.FloatObject]

> Read-only property accessing the right horizontal coordinate.

property top: Optional[pypdf.generic._base.FloatObject]

> Read-only property accessing the top vertical coordinate.

property bottom: Optional[pypdf.generic._base.FloatObject]

> Read-only property accessing the bottom vertical coordinate.

property color: Optional[pypdf.generic._data_structures.ArrayObject]

> Read-only property accessing the color in (R, G, B) with values 0.0-1.0.

property font_format: Optional[pypdf.constants.OutlineFontFlag]

> Read-only property accessing the font type.
>
> 1=italic, 2=bold, 3=both

property outline_count: Optional[int]

> Read-only property accessing the outline count.
>
> positive = expanded negative = collapsed absolute value = number of visible descendants at all levels

THE RECTANGLEOBJECT CLASS

class pypdf.generic.**RectangleObject**(*arr: Union[RectangleObject, Tuple[float, float, float, float]]*)

Bases: `ArrayObject`

This class is used to represent *page boxes* in pypdf.

These boxes include:

- *artbox*
- *bleedbox*
- *cropbox*
- *mediabox*
- *trimbox*

scale(*sx: float*, *sy: float*) → *RectangleObject*

ensureIsNumber(*value: Any*) → Union[FloatObject, NumberObject]

property left: pypdf.generic._base.FloatObject

property bottom: pypdf.generic._base.FloatObject

property right: pypdf.generic._base.FloatObject

property top: pypdf.generic._base.FloatObject

getLowerLeft_x() → FloatObject

getLowerLeft_y() → FloatObject

getUpperRight_x() → FloatObject

getUpperRight_y() → FloatObject

getUpperLeft_x() → FloatObject

getUpperLeft_y() → FloatObject

getLowerRight_x() → FloatObject

getLowerRight_y() → FloatObject

property lower_left: Tuple[float, float]

    Property to read and modify the lower left coordinate of this box in (x,y) form.

property lower_right: Tuple[float, float]

    Property to read and modify the lower right coordinate of this box in (x,y) form.

property upper_left: Tuple[float, float]

    Property to read and modify the upper left coordinate of this box in (x,y) form.

property upper_right: Tuple[float, float]

    Property to read and modify the upper right coordinate of this box in (x,y) form.

getLowerLeft() → Tuple[float, float]

getLowerRight() → Tuple[float, float]

getUpperLeft() → Tuple[float, float]

getUpperRight() → Tuple[float, float]

setLowerLeft(*value: Tuple[float, float]*) → None

setLowerRight(*value: Tuple[float, float]*) → None

setUpperLeft(*value: Tuple[float, float]*) → None

setUpperRight(*value: Tuple[float, float]*) → None

property width: float

getWidth() → float

property height: float

getHeight() → float

property lowerLeft: Tuple[float, float]

property lowerRight: Tuple[float, float]

property upperLeft: Tuple[float, float]

property upperRight: Tuple[float, float]

THE FIELD CLASS

class pypdf.generic.**Field**(*data: DictionaryObject*)

Bases: `TreeObject`

A class representing a field dictionary.

This class is accessed through *get_fields()*

property field_type: Optional[pypdf.generic._base.NameObject]
Read-only property accessing the type of this field.

property fieldType: Optional[pypdf.generic._base.NameObject]
Use *field_type* instead.

Deprecated since version 1.28.3.

property parent: Optional[pypdf.generic._data_structures.DictionaryObject]
Read-only property accessing the parent of this field.

property kids: Optional[pypdf.generic._data_structures.ArrayObject]
Read-only property accessing the kids of this field.

property name: Optional[str]
Read-only property accessing the name of this field.

property alternate_name: Optional[str]
Read-only property accessing the alternate name of this field.

property altName: Optional[str]
Use *alternate_name* instead.

Deprecated since version 1.28.3.

property mapping_name: Optional[str]
Read-only property accessing the mapping name of this field.

This name is used by pypdf as a key in the dictionary returned by *get_fields()*

property mappingName: Optional[str]
Use *mapping_name* instead.

Deprecated since version 1.28.3.

property flags: Optional[int]
Read-only property accessing the field flags, specifying various characteristics of the field (see Table 8.70 of the PDF 1.7 reference).

property value: Optional[Any]

> Read-only property accessing the value of this field.
>
> Format varies based on field type.

property default_value: Optional[Any]

> Read-only property accessing the default value of this field.

property defaultValue: Optional[Any]

> Use *default_value* instead.
>
> Deprecated since version 1.28.3.

property additional_actions:
Optional[pypdf.generic._data_structures.DictionaryObject]

> Read-only property accessing the additional actions dictionary.
>
> This dictionary defines the field's behavior in response to trigger events. See Section 8.5.2 of the PDF 1.7 reference.

property additionalActions:
Optional[pypdf.generic._data_structures.DictionaryObject]

> Use *additional_actions* instead.
>
> Deprecated since version 1.28.3.

THE PAGERANGE CLASS

class pypdf.**PageRange**(*arg: Union[slice, PageRange, str]*)

Bases: object

A slice-like representation of a range of page indices.

For example, page numbers, only starting at zero.

The syntax is like what you would put between brackets [ ]. The slice is one of the few Python types that can't be subclassed, but this class converts to and from slices, and allows similar use.

- PageRange(str) parses a string representing a page range.

- PageRange(slice) directly "imports" a slice.

- to_slice() gives the equivalent slice.

- str() and repr() allow printing.

- indices(n) is like slice.indices(n).

static valid(*input: Any*) → bool

True if input is a valid initializer for a PageRange.

> **Parameters input** – A possible PageRange string or a PageRange object.
>
> **Returns** True, if the input is a valid PageRange.

to_slice() → slice

Return the slice equivalent of this page range.

indices(*n: int*) → Tuple[int, int, int]

Assuming a sequence of length n, calculate the start and stop indices, and the stride length of the PageRange.

See help(slice.indices).

> **Parameters n** – the length of the list of pages to choose from.
>
> **Returns** Arguments for range()

THE ANNOTATIONS MODULE

PDF specifies several annotation types which pypdf makes available here.

The names of the annotations and their attributes do not reflect the names in the specification in all cases. For example, the PDF standard defines a 'Square' annotation that does not actually need to be square. For this reason, pypdf calls it 'Rectangle'.

At their core, all annotation types are DictionaryObjects. That means if pypdf does not implement a feature, users can easily extend the given functionality.

class pypdf.annotations.**AnnotationDictionary**

> Bases: DictionaryObject, ABC
>
> **property flags: pypdf.constants.AnnotationFlag**

class pypdf.annotations.**MarkupAnnotation**(**, title_bar: Optional[str] = None*)

> Bases: *AnnotationDictionary*, ABC
>
> Base class for all markup annotations.
>
> > **Parameters** title_bar – Text to be displayed in the title bar of the annotation; by convention this is the name of the author

class pypdf.annotations.**Ellipse**(*rect: Union[RectangleObject, Tuple[float, float, float, float]], *, interiour_color: Optional[str] = None, **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**FreeText**(**, text: str, rect: Union[RectangleObject, Tuple[float, float, float, float]], font: str = 'Helvetica', bold: bool = False, italic: bool = False, font_size: str = '14pt', font_color: str = '000000', border_color: Optional[str] = '000000', background_color: Optional[str] = 'ffffff', **kwargs: Any*)

> Bases: *MarkupAnnotation*
>
> A FreeText annotation

class pypdf.annotations.**Highlight**(**, rect: Union[RectangleObject, Tuple[float, float, float, float]], quad_points: ArrayObject, highlight_color: str = 'ff0000', **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**Line**(*p1: Tuple[float, float], p2: Tuple[float, float], rect: Union[RectangleObject, Tuple[float, float, float, float]], text: str = '', **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**Link**(*, *rect: ~typing.Union[~pypdf.generic._rectangle.RectangleObject,*
    ~typing.Tuple[float, float, float, float]], border:
    ~typing.Optional[~pypdf.generic._data_structures.ArrayObject] = None, url:
    ~typing.Optional[str] = None, target_page_index: ~typing.Optional[int] =
    *None, fit: ~pypdf.generic._fit.Fit = <pypdf.generic._fit.Fit object>, **kwargs:*
    ~typing.Any)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**Polygon**(*vertices: List[Tuple[float, float]], **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**PolyLine**(*vertices: List[Tuple[float, float]], **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**Rectangle**(*rect: Union[RectangleObject, Tuple[float, float, float, float]], *,*
    *interiour_color: Optional[str] = None, **kwargs: Any*)

> Bases: *MarkupAnnotation*

class pypdf.annotations.**Text**(*, *rect: ~typing.Union[~pypdf.generic._rectangle.RectangleObject,*
    ~typing.Tuple[float, float, float, float]], text: str, open: bool = False, flags: int
    *= AnnotationFlag.None, **kwargs: ~typing.Any*)

> Bases: *MarkupAnnotation*

> A text annotation.

> > **Parameters**
> >
> > - **rect** – array of four integers [xLL, yLL, xUR, yUR] specifying the clickable rectangular
> >   area
> >
> > - **text** – The text that is added to the document
> >
> > - **open** –
> >
> > - **flags** –

class pypdf.annotations.**Popup**(*, *rect: Union[RectangleObject, Tuple[float, float, float, float]], parent:*
    *Optional[DictionaryObject] = None, open: bool = False, **kwargs: Any*)

> Bases: *AnnotationDictionary*

THE FIT CLASS

class pypdf.generic.**Fit**(*fit_type: str*, *fit_args: Tuple[Union[None, float, Any], ...] = ()*)

    Bases: object

    classmethod xyz(*left: Optional[float] = None*, *top: Optional[float] = None*, *zoom: Optional[float] = None*) → *Fit*

        Display the page designated by page, with the coordinates (left , top) positioned at the upper-left corner of the window and the contents of the page magnified by the factor zoom.

        A null value for any of the parameters left, top, or zoom specifies that the current value of that parameter is to be retained unchanged.

        A zoom value of 0 has the same meaning as a null value.

        Parameters

            • **left** –

            • **top** –

            • **zoom** –

        Returns The created fit object.

    classmethod fit() → *Fit*

        Display the page designated by page, with its contents magnified just enough to fit the entire page within the window both horizontally and vertically.

        If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the page within the window in the other dimension.

    classmethod fit_horizontally(*top: Optional[float] = None*) → *Fit*

        Display the page designated by page , with the vertical coordinate top positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of the page within the window.

        A null value for top specifies that the current value of that parameter is to be retained unchanged.

        Parameters top –

        Returns The created fit object.

    classmethod fit_vertically(*left: Optional[float] = None*) → *Fit*

    classmethod fit_rectangle(*left: Optional[float] = None*, *bottom: Optional[float] = None*, *right: Optional[float] = None*, *top: Optional[float] = None*) → *Fit*

        Display the page designated by page , with its contents magnified just enough to fit the rectangle specified by the coordinates left, bottom, right, and top entirely within the window both horizontally and vertically.

If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the rectangle within the window in the other dimension.

A null value for any of the parameters may result in unpredictable behavior.

> **Parameters**
>
> > - **left** –
> >
> > - **bottom** –
> >
> > - **right** –
> >
> > - **top** –
>
> **Returns** The created fit object.

classmethod fit_box() → *Fit*

> Display the page designated by page , with its contents magnified just enough to fit its bounding box entirely within the window both horizontally and vertically.
>
> If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the bounding box within the window in the other dimension.

classmethod fit_box_horizontally(*top: Optional[float] = None*) → *Fit*

> Display the page designated by page , with the vertical coordinate top positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of its bounding box within the window.
>
> A null value for top specifies that the current value of that parameter is to be retained unchanged.
>
> > **Parameters top** –
>
> **Returns** The created fit object.

classmethod fit_box_vertically(*left: Optional[float] = None*) → *Fit*

> Display the page designated by page, with the horizontal coordinate left positioned at the left edge of the window and the contents of the page magnified just enough to fit the entire height of its bounding box within the window.
>
> A null value for left specifies that the current value of that parameter is to be retained unchanged.
>
> > **Parameters left** –
>
> **Returns** The created fit object.

THE PAPERSIZE CLASS

class pypdf.**PaperSize**

Bases: object

(width, height) of the paper in portrait mode in pixels at 72 ppi.

A0 = Dimensions(width=2384, height=3370)

A1 = Dimensions(width=1684, height=2384)

A2 = Dimensions(width=1191, height=1684)

A3 = Dimensions(width=842, height=1191)

A4 = Dimensions(width=595, height=842)

A5 = Dimensions(width=420, height=595)

A6 = Dimensions(width=298, height=420)

A7 = Dimensions(width=210, height=298)

A8 = Dimensions(width=147, height=210)

C4 = Dimensions(width=649, height=918)

38.1 Add blank page with PaperSize

```python
from pypdf import PaperSize, PdfReader, PdfWriter

pdf_reader = PdfReader("sample.pdf")
pdf_writer = PdfWriter()
pdf_writer.append_pages_from_reader(pdf_reader)
pdf_writer.add_blank_page(PaperSize.A8.width, PaperSize.A8.height)
with open("output.pdf", "wb") as output_stream:
    pdf_writer.write(output_stream)
```

38.2 Insert blank page with PaperSize

```python
from pypdf import PaperSize, PdfReader, PdfWriter

pdf_reader = PdfReader("sample.pdf")
pdf_writer = PdfWriter()
pdf_writer.append_pages_from_reader(pdf_reader)
pdf_writer.insert_blank_page(PaperSize.A8.width, PaperSize.A8.height, 1)
with open("output.pdf", "wb") as output_stream:
    pdf_writer.write(output_stream)
```

DEVELOPER INTRO

pypdf is a library and hence its users are developers. This document is not for the users, but for people who want to work on pypdf itself.

39.1 Installing Requirements

```
pip install -r requirements/dev.txt
```

39.2 Running Tests

See *testing pypdf with pytest*

39.3 The sample-files git submodule

The reason for having the submodule `sample-files` is that we want to keep the size of the pypdf repository small while we also want to have an extensive test suite. Those two goals contradict each other.

The `resources` folder should contain a select set of core examples that cover most cases we typically want to test for. The `sample-files` might cover a lot more edge cases, the behavior we get when file sizes get bigger, different PDF producers.

In order to get the sample-files folder, you need to execute:

```
git submodule update --init
```

39.4 Tools: git and pre-commit

Git is a command line application for version control. If you don't know it, you can play ohmygit to learn it.

GitHub is the service where the pypdf project is hosted. While git is free and open source, GitHub is a paid service by Microsoft - but for free in lot of cases.

pre-commit is a command line application that uses git hooks to automatically execute code. This allows you to avoid style issues and other code quality issues. After you entered `pre-commit install` once in your local copy of pypdf, it will automatically be executed when you `git commit`.

39.5 Commit Messages

Having a clean commit message helps people to quickly understand what the commit was about, without actually looking at the changes. The first line of the commit message is used to auto-generate the CHANGELOG. For this reason, the format should be:

```
PREFIX: DESCRIPTION

BODY
```

The `PREFIX` can be:

- `BUG`: A bug was fixed. Likely there is one or multiple issues. Then write in the `BODY`: `Closes #123` where 123 is the issue number on GitHub. It would be absolutely amazing if you could write a regression test in those cases. That is a test that would fail without the fix.
- `ENH`: A new feature! Describe in the body what it can be used for.
- `DEP`: A deprecation - either marking something as "this is going to be removed" or actually removing it.
- `PI`: A performance improvement. This could also be a reduction in the file size of PDF files generated by pypdf.
- `ROB`: A robustness change. Dealing better with broken PDF files.
- `DOC`: A documentation change.
- `TST`: Adding / adjusting tests.
- `DEV`: Developer experience improvements - e.g. pre-commit or setting up CI
- `MAINT`: Quite a lot of different stuff. Performance improvements are for sure the most interesting changes in here. Refactorings as well.
- `STY`: A style change. Something that makes pypdf code more consistent. Typically a small change.

39.6 Benchmarks

We need to keep an eye on performance and thus we have a few benchmarks.

See py-pdf.github.io/pypdf/dev/bench

THE PDF FORMAT

It's recommended to look in the PDF specification for details and clarifications. This is only intended to give a very rough overview of the format.

40.1 Overall Structure

A PDF consists of:

1. Header: Contains the version of the PDF, e.g. `%PDF-1.7`

2. Body: Contains a sequence of indirect objects

3. Cross-reference table (xref): Contains a list of the indirect objects in the body

4. Trailer

40.2 The xref table

A cross-reference table (xref) is a table of the indirect objects in the body. It allows quick access to those objects by pointing to their location in the file.

It looks like this:

```
xref 42 5
0000001000 65535 f
0000001234 00000 n
0000001987 00000 n
0000011987 00000 n
0000031987 00000 n
```

Let's go through it step-by-step:

- `xref` is just a keyword that specifies the start of the xref table.

- `42` is the numerical ID of the first object in this xref section; `5` is the number of entries in the xref table.

- Now every object has 3 entries `nnnnnnnnnn ggggg n`: The 10-digit byte offset, a 5-digit generation number, and a literal keyword which is either `n` or `f`.

    - `nnnnnnnnnn` is the byte offset of the object. It tells the reader where the object is in the file.

    - `ggggg` is the generation number. It tells the reader how old the object is.

    - `n` means that the object is a normal in-use object, `f` means that the object is a free object.

* The first free object always has a generation number of 65535. It forms the head of a linked-list of all free objects.

* The generation number of a normal object is always 0. The generation number allows the PDF format to contain multiple versions of the same object. This is a version history mechanism.

40.3 The body

The body is a sequence of indirect objects:

```
counter generationnumber << the_object >> endobj
```

* `counter` (integer) is a unique identifier for the object.

* `generationnumber` (integer) is the generation number of the object.

* `the_object` is the object itself. It can be empty. Starts with `/Keyword` to specify which kind of object it is.

* `endobj` marks the end of the object.

A concrete example can be found in `test_reader.py::test_get_images_raw`:

```
1 0 obj << /Count 1 /Kids [4 0 R] /Type /Pages >> endobj
2 0 obj << >> endobj
3 0 obj << >> endobj
4 0 obj << /Contents 3 0 R /CropBox [0.0 0.0 2550.0 3508.0]
 /MediaBox [0.0 0.0 2550.0 3508.0] /Parent 1 0 R
 /Resources << /Font << >> >>
 /Rotate 0 /Type /Page >> endobj
5 0 obj << /Pages 1 0 R /Type /Catalog >> endobj
```

40.4 The trailer

The trailer looks like this:

```
trailer << /Root 5 0 R
           /Size 6
        >>
startxref 1234
%%EOF
```

Let's go through it:

* `trailer <<` indicates that the *trailer dictionary* starts. It ends with `>>`.

* `startxref` is a keyword followed by the byte-location of the `xref` keyword. As the trailer is always at the bottom of the file, this allows readers to quickly find the xref table.

* `%%EOF` is the end-of-file marker.

The trailer dictionary is a key-value list. The keys are specified in Table 3.13 of the PDF Reference 1.7, e.g. `/Root` and `/Size` (both are required).

* `/Root` (dictionary) contains the document catalog.

    – The 5 is the object number of the catalog dictionary

- **0** is the generation number of the catalog dictionary

  - **R** is the keyword that indicates that the object is a reference to the catalog dictionary.

- **/Size** (integer) contains the total number of entries in the files xref table.

40.5 Reading PDF files

Most PDF files are compressed. If you want to read them, first uncompress them:

```
pdftk crazyones.pdf output crazyones-uncomp.pdf uncompress
```

Then rename `crazyones-uncomp.pdf` to `crazyones-uncomp.txt` and open it in our favorite IDE / text editor.

HOW PYPDF PARSES PDF FILES

pypdf uses *PdfReader* to parse PDF files. The method *PdfReader.read* shows the basic structure of parsing:

1. **Finding and reading the cross-reference tables / trailer**: The cross-reference table (xref table) is a table of byte offsets that indicate the locations of objects within the file. The trailer provides additional information such as the root object (Catalog) and the Info object containing metadata.

2. **Parsing the objects**: After locating the xref table and the trailer, pypdf proceeds to parse the objects in the PDF. Objects in a PDF can be of various types such as dictionaries, arrays, streams, and simple data types (e.g., integers, strings). pypdf parses these objects and stores them in `PdfReader.resolved_objects` via *cache_indirect_object*.

3. **Decoding content streams**: The content of a PDF is typically stored in content streams, which are sequences of PDF operators and operands. pypdf decodes these content streams by applying filters (e.g., `FlateDecode`, `LZWDecode`) specified in the stream's dictionary. This is only done when the object is requested via *PdfReader.get_object* in the `PdfReader._get_object_from_stream` method.

41.1 References

PDF 1.7 specification:

- 7.5 File Structure
- 7.5.4 Cross-Reference Table
- 7.8 Content Streams and Resources

HOW PYPDF WRITES PDF FILES

pypdf uses *PdfWriter* to write PDF files. pypdf has `PdfObject` and several subclasses with the `write_to_stream` method. The *PdfWriter.write* method uses the `write_to_stream` methods of the referenced objects.

The *PdfWriter.write_stream* method has the following core steps:

1. `_sweep_indirect_references`: This step ensures that any circular references to objects are correctly handled. It adds the object reference numbers of any circularly referenced objects to an external reference map, so that self-page-referencing trees can reference the correct new object location, rather than copying in a new copy of the page object.

2. **Write the File Header and Body** with `_write_pdf_structure`: In this step, the PDF header and objects are written to the output stream. This includes the PDF version (e.g., %PDF-1.7) and the objects that make up the content of the PDF, such as pages, annotations, and form fields. The locations (byte offsets) of these objects are stored for later use in generating the xref table.

3. **Write the Cross-Reference Table** with `_write_xref_table`: Using the stored object locations, this step generates and writes the cross-reference table (xref table) to the output stream. The cross-reference table contains the byte offsets for each object in the PDF file, allowing for quick random access to objects when reading the PDF.

4. **Write the File Trailer** with `_write_trailer`: The trailer is written to the output stream in this step. The trailer contains essential information, such as the number of objects in the PDF, the location of the root object (Catalog), and the Info object containing metadata. The trailer also specifies the location of the xref table.

42.1 How others do it

Looking at altrnative software designs and implementations can help to improve our choices.

42.1.1 fpdf2

fpdf2 has a PDFObject class with a serialize method which roughly maps to `pypdf.PdfObject.write_to_stream`. Some other similarities include:

- fpdf.output.OutputProducer.buffersize vs *pypdf.PdfWriter.write_stream*

- fpdpf.syntax.Name vs `pypdf.generic.NameObject`

- fpdf.syntax.build_obj_dict vs `pypdf.generic.DictionaryObject`

- fpdf.structure_tree.NumberTree vs `pypdf.generic.TreeObject`

42.1.2 pdfrw

pdfrw, in contrast, seems to work more with the standard Python objects (bool, float, string) and not wrap them in custom objects, if possible. It still has:

- PdfArray

- PdfDict

- PdfName

- PdfString

- PdfIndirect

The core classes of pdfrw are PdfReader and PdfWriter

CMAPS

Looking at the cmap of "crazyones":

```
pdftk crazyones.pdf output crazyones-uncomp.pdf uncompress
```

You can see this:

```
begincmap
/CMapName /T1Encoding-UTF16 def
/CMapType 2 def
/CIDSystemInfo <<
  /Registry (Adobe)
  /Ordering (UCS)
  /Supplement 0
>> def
1 begincodespacerange
<00> <FF>
endcodespacerange
1 beginbfchar
<1B> <FB00>
endbfchar
endcmap
CMapName currentdict /CMap defineresource pop
```

43.1 codespacerange

A codespacerange maps a complete sequence of bytes to a range of unicode glyphs. It defines a starting point:

```
1 beginbfchar
<1B> <FB00>
```

That means that 1B (Hex for 27) maps to the unicode character FB00 - the ligature ff (two lowercase f's).

The two numbers in `begincodespacerange` mean that it starts with an offset of 0 (hence from 1B FB00) upt to an offset of FF (dec: 255), hence 1B+FF = 282 FBFF.

Within the text stream, there is

```
(The)-342(mis\034ts.)
```

\034 is octal for 28 decimal.

THE DEPRECATION PROCESS

pypdf strives to be an excellent library for its current users and for new ones. We are careful with introducing potentially breaking changes, but we will do them if they provide value for the community on the long run.

We hope and think that deprecations will not happen soon again. If they do, users can rely on the following procedure.

44.1 Semantic Versioning

pypdf uses semantic versioning. If you want to avoid breaking changes, please use dependency pinning (also known as version pinning). In Python, this is done by specifying the exact version you want to use in a `requirements.txt` file. A tool that can support you is `pip-compile` from `pip-tools`.

If you are using Poetry it is done with the `poetry.lock` file.

44.2 How pypdf deprecates features

Assume the current version of pypdf is `x.y.z`. After a discussion (e.g. via GitHub issues) we decided to remove a class / function / method. This is how we do it:

1. `x.y.(z+1)`: Add a DeprecationWarning. If there is a replacement, the replacement is also introduced and the warning informs about the change and when it will happen. The docs let users know about the deprecation and when it will happen and the new function. The CHANGELOG informs about it.

2. `(x+1).0.0`: Remove / change the code in the breaking way by replacing DeprecationWarnings by Deprecation-Errors. We do this to help people who didn't look at the warnings before. The CHANGELOG informs about it.

3. `(x+2).0.0`: The DeprecationErrors are removed.

This means the users have 3 warnings in the CHANGELOG, a DeprecationWarning until the next major release and a DeprecationError until the major release after that.

Please note that adding warnings can be a breaking change for some users; most likely just in the CI. This means it needs to be properly documented.

DOCUMENTATION

45.1 API Reference

45.1.1 Method / Function Docstrings

We use Google-Style Docstrings:

```python
def example(param1: int, param2: str) -> bool:
    """
    Example function with PEP 484 type annotations.

    Args:
      param1: The first parameter.
      param2: The second parameter.

    Returns:
      The return value. True for success, False otherwise.

    Raises:
      AttributeError: The ``Raises`` section is a list of all exceptions
        that are relevant to the interface.
      ValueError: If `param2` is equal to `param1`.

    Examples:
        Examples should be written in doctest format, and should illustrate how
        to use the function.

        >>> print([i for i in example_generator(4)])
        [0, 1, 2, 3]
    """
```

- The order of sections is (1) Args (2) Returns (3) Raises (4) Examples

- If there is no return value, remove the 'Returns' block

- Properties should not have any sections

45.2 Issues and PRs

An issue can be used to discuss what we want to achieve.

A PR can be used to discuss how we achieve it.

45.3 Commit Messages

We want to have descriptive commits in the `main` branch. For this reason, every pull request (PR) is squashed. That means no matter how many commits a PR has, in the end only one combined commit will be in `main`.

The title of the PR will be used as the first line of that combined commit message.

The first comment within the commit will be used as the message body.

See dev intro for more details.

TESTING

pypdf uses `pytest` for testing.

To run the tests you need to install the CI (Continuous Integration) requirements by running `pip install -r requirements/ci.txt` or `pip install -r requirements/ci-3.11.txt` if running python 3.11.

46.1 De-selecting groups of tests

pypdf makes use of the following pytest markers:

- `slow`: Tests that require more than 5 seconds
- `samples`: Tests that require the the `sample-files` git submodule to be initialized. As of October 2022, this is about 25 MB.
- `enable_socket`: Tests that download PDF documents. They are stored locally and thus only need to be downloaded once. As of October 2022, this is about 200 MB.

You can disable them by `pytest -m "not enable_socket"` or `pytest -m "not samples"`. You can even disable all of them: `pytest -m "not enable_socket" -m "not samples" -m "not slow"`.

Please note that this reduces test coverage. The CI will always test all files.

46.2 Creating a Coverage Report

If you want to get a coverage report that considers the Python version specific code, you can run `tox`.

As a prerequisite, we recommend using `pyenv` so that you can install the different Python versions:

```
pyenv install pypy3.8-7.3.7
pyenv install 3.6.15
pyenv install 3.7.12
pyenv install 3.8.12
pyenv install 3.9.10
pyenv install 3.10.2
```

Then you can execute `tox` which will create a coverage report in HTML form in the end. The execution takes about 30 minutes.

46.3 Docstrings in Unit tests

The first line of a docstring in a unit test should be written in a way that you could prefix it with "This tests ensures that …", e.g.

- Invalid XML in xmp_metadata is gracefully handled.

- The identity is returning its input.

- xmp_modify_date is extracted correctly.

This way, plugins like `pytest-testdox` can generate really nice output when the tests are running. This looks similar to the output of mocha.js.

If the test is a regression test, write

> This test is a regression test for issue #1234

If the regression test is just one parameter of other tests, then add it as a comment for that parameter.

46.4 Evaluate a PR in-progress version

You may want to test a version from a PR which has not been released yet. The easiest way is to use pip and install a version from git:

a) Go the PR and identify the repository and branch.
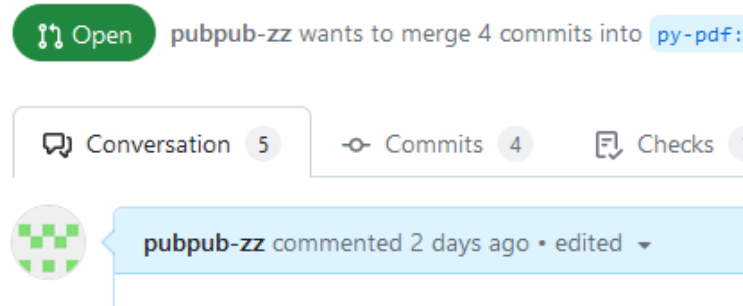
Example from below : repository: **pubpub-zz** / branch: **iss2200** :

b) you can then install the version using pip from git:

Example:

```
pip install git+https://github.com/pubpub-zz/pypdf.git@iss2200
```

RELEASING

A `pypdf` release contains the following artifacts:

- A new release on PyPI

- A release commit

    - Containing a changelog update

    - A new git tag

        * A Github release

47.1 Who does it?

`pypdf` should typically only be released by one of the core maintainers / the core maintainer. At the moment, this is Martin Thoma.

Any owner of the py-pdf organization has also the technical permissions to release.

47.2 How is it done?

The release contains the following steps:

1. Update the CHANGELOG.md and the _version.py via `python make_release.py`. This also prepares the release commit message

2. Create a release commit: `git commit -eF RELEASE_COMMIT_MSG.md`.

3. Tag that commit: `git tag -s {{version}} -eF RELEASE_TAG_MSG.md`.

4. Push both: `git push && git push --tags`

5. CI now builds a source and a wheels package which it pushes to PyPI. It also creates a GitHub release.

```
      python
   make_release.py
         │
         ▼
   Manually adjust
   CHANGELOG.md
      changes
         │
         ▼
   Is there a breaking
        change
```

Semantic
Versioning

```
   Yes          No

Major version bump      Is there a new
  in _version.py           feature?

                            Yes

                    Minor version bump      Patch version bump
```

Adjust the
CHANGELOG.md

git commit -eF RELEASE_COMMIT_MSG.md

git tag -s {{version}} -eF RELEASE_TAG_MSG.md

Add the CHANGELOG changes in the body

Github Action

Build and push
packages to PyPI

Create release on
Github

47.2.1 The Release Tag

- Use the release version as the tag name. No need for a leading "v".

- Use the changelog entry as the body

47.3 When are releases done?

There is no need to wait for anything. If the CI is green (all tests succeeded), we can release.

I (Martin Thoma) typically only release once a week, because it costs a little bit of time and I don't want to spam users with too many releases.

CHANGELOG

48.1 Version 3.17.0, 2023-10-29

48.1.1 Security (SEC)

- Infinite recursion when using PdfWriter(clone_from=reader) (#2264)

48.1.2 New Features (ENH)

- Add parameter to select images to be removed (#2214)

48.1.3 Bug Fixes (BUG)

- Correctly handle image mode 1 with FlateDecode (#2249)
- Error when filling a value with parentheses #2268 (#2269)
- Handle empty root outline (#2239)

Full Changelog

48.2 Version 3.16.4, 2023-10-10

48.2.1 Bug Fixes (BUG)

- Avoid exceeding recursion depth when retrieving image mode (#2251)

Full Changelog

48.3 Version 3.16.3, 2023-10-08

48.3.1 Bug Fixes (BUG)

- Invalid cm/tm in visitor functions (#2206)
- Encrypt / decrypt Stream object dictionaries (#2228)
- Support nested color spaces for the /DeviceN color space (#2241)
- Images property fails if NullObject in list (#2215)

48.3.2 Developer Experience (DEV)

- Unify mypy options and warn redundant workarounds (#2223)

Full Changelog

48.4 Version 3.16.2, 2023-09-24

48.4.1 Bug Fixes (BUG)

- PDF size increases because of too high float writing precision (#2213)
- Fix test_watermarking_reportlab_rendering() (#2203)

Full Changelog

48.5 Version 3.16.1, 2023-09-17

The 'rename PdfWriter.create_viewer_preference to PdfWriter.create_viewer_preferences (#2190)' could be a breaking change for you, if you use it. As it was only introduced last week I'm confident enough that nobody will be affected though. Hence only the patch update.

48.5.1 Bug Fixes (BUG)

- Missing new line in extract_text with cm operations (#2142)
- _get_fonts not processing properly CIDFonts and annotations (#2194)

48.5.2 Maintenance (MAINT)

- Rename PdfWriter.create_viewer_preference to PdfWriter.create_viewer_preferences (#2190)

Full Changelog

48.6 Version 3.16.0, 2023-09-10

48.6.1 Security (SEC)

- Infinite recursion caused by IndirectObject clone (#2156)

48.6.2 New Features (ENH)

- Ease access to ViewerPreferences (#2144)

48.6.3 Bug Fixes (BUG)

- Catch the case where w[0] is an IndirectObject instead of an int (#2154)
- Cope with indirect objects in filters and remove deprecated code (#2177)
- Accept tabs in cmaps (#2174) / cope with extra space (#2151)
- Merge pages without resources (#2150)
- getcontents() shall return None if contents is NullObject (#2161)
- Fix conversion from 1 to LA (#2175)

48.6.4 Robustness (ROB)

- Accept XYZ with no arguments (#2178)

Full Changelog

48.7 Version 3.15.5, 2023-09-03

48.7.1 Bug Fixes (BUG)

- Cope with missing /I in articles (#2134)
- Fix image look-up table in EncodedStreamObject (#2128)
- remove_images not operating in sub level forms (#2133)

48.7.2 Robustness (ROB)

- Cope with damaged PDF (#2129)

Full Changelog

48.8 Version 3.15.4, 2023-08-27

48.8.1 Performance Improvements (PI)

- Making pypdf as fast as pdfrw (#2086)

48.8.2 Maintenance (MAINT)

- Relax typing_extensions version (#2104)

Full Changelog

48.9 Version 3.15.3, 2023-08-26

48.9.1 Bug Fixes (BUG)

- Check version of crypt provider (#2115)
- TypeError: can't concat str to bytes (#2114)
- Require flit_core >= 3.9 (#2091)

Full Changelog

48.10 Version 3.15.2, 2023-08-20

48.10.1 Security (SEC)

- Avoid endless recursion of reading damaged PDF file (#2093)

48.10.2 Performance Improvements (PI)

- Re-use content stream (#2101)

48.10.3 Maintenance (MAINT)

- Make ParseError inherit from PyPdfError (#2097)

Full Changelog

48.11 Version 3.15.1, 2023-08-13

48.11.1 Performance Improvements (PI)

- optimize _decode_png_prediction (#2068)

48.11.2 Bug Fixes (BUG)

- Fix incorrect tm_matrix in call to visitor_text (#2060)
- Writing German characters into form fields (#2047)
- Prevent stall when accessing image in corrupted pdf (#2081)
- append() fails when articles do not have /T (#2080)

48.11.3 Robustness (ROB)

- Cope with xref not followed by separator (#2083)

Full Changelog

48.12 Version 3.15.0, 2023-08-06

48.12.1 New Features (ENH)

- Add `level` parameter to compress_content_streams (#2044)
- Process /uniHHHH for text_extract (#2043)

48.12.2 Bug Fixes (BUG)

- Fix AnnotationBuilder.link (#2066)
- JPX image without ColorSpace (#2062)
- Added check for field /Info when cloning reader document (#2055)
- Fix indexed/CMYK images (#2039)

48.12.3 Maintenance (MAINT)

- Cryptography as primary dependency (#2053)

Full Changelog

48.13 Version 3.14.0, 2023-07-29

48.13.1 New Features (ENH)

- Accelerate image list keys generation (#2014)
- Use `cryptography` for encryption/decryption as a fallback for PyCryptodome (#2000)
- Extract LaTeX characters (#2016)
- ASCIIHexDecode.decode now returns bytes instead of str (#1994)

48.13.2 Bug Fixes (BUG)

- Add RunLengthDecode filter (#2012)
- Process /Separation ColorSpace (#2007)
- Handle single element ColorSpace list (#2026)
- Process lookup decoded as TextStringObjects (#2008)

48.13.3 Robustness (ROB)

- Cope with garbage collector during cloning (#1841)

48.13.4 Maintenance (MAINT)

- Cleanup of annotations (#1745)

Full Changelog

48.14 Version 3.13.0, 2023-07-23

48.14.1 New Features (ENH)

- Add is_open in outlines in PdfReader and PdfWriter (#1960)

48.14.2 Bug Fixes (BUG)

- Search /DA in hierarchy fields (#2002)
- Cope with different ISO date length (#1999)
- Decode Black only/CMYK deviceN images (#1984)
- Process CMYK in deflate images (#1977)

48.14.3 Developer Experience (DEV)

- Add mypy to pre-commit (#2001)
- Release automation (#1991, #1985)

Full Changelog

48.15 Version 3.12.2, 2023-07-16

48.15.1 Bug Fixes (BUG)

- Accept calRGB and calGray color_spaces (#1968)
- Process 2bits and 4bits images (#1967)
- Check for AcroForm and ensure it is not None (#1965)

48.15.2 Developer Experience (DEV)

- Automate the release process (#1970)

Full Changelog

48.16 Version 3.12.1, 2023-07-09

48.16.1 Bug Fixes (BUG)

- Prevent updating page contents after merging page (stamping/watermarking) (#1952)

-

- Inverse color in CMYK images (#1947)
- Dates conversion not working with Z00'00' (#1946)
- Support UTF-16-LE Strings (#1884)

Full Changelog

48.17 Version 3.12.0, 2023-07-02

48.17.1 New Features (ENH)

- Add AES support for encrypting PDF files (#1918, #1935, #1936, #1938)
- Add page deletion feature to PdfWriter (#1843)

48.17.2 Bug Fixes (BUG)

- PdfReader.get_fields() attempts to delete non-existing index "/Off" (#1933)
- Remove unused objects when cloning_from (#1926)
- Add the TK.SIZE into the trailer (#1911)
- add_named_destination() maintains named destination list sort order (#1930)

Full Changelog

48.18 Version 3.11.1, 2023-06-25

48.18.1 Bug Fixes (BUG)

- Cascaded filters in image objects (#1913)
- Append pdf with named destination using numbers for pages (#1858)
- Ignore "/B" fields only on pages in PdfWriter.append() (#1875)

Full Changelog

48.19 Version 3.11.0, 2023-06-23

48.19.1 New Features (ENH)

- Add page_number property (#1856)

48.19.2 Bug Fixes (BUG)

- File expansion when updating with Page Contents (#1906)
- Missing Alternate in indexed/ICCbased colorspaces (#1896)

Full Changelog

48.20 Version 3.10.0, 2023-06-18

48.20.1 New Features (ENH)

- Extraction of inline images (#1850)
- Add capability to replace image (#1849)
- Extend images interface by returning an ImageFile(File) class (#1848)
- Add set_data to EncodedStreamObject (#1854)

48.20.2 Bug Fixes (BUG)

- Fix RGB FlateEncode Images(PNG) and transparency (#1834)
- Generate static appearance for fields (#1864)

Full Changelog

48.21 Version 3.9.1, 2023-06-04

48.21.1 Deprecations (DEP)

- Deprecate PdfMerger (#1866)

48.21.2 Bug Fixes (BUG)

- Ignore UTF-8 decode errors (#1865)

48.21.3 Robustness (ROB)

- Handle missing /Type entry in Page tree (#1859)

Full Changelog

48.22 Version 3.9.0, 2023-05-21

48.22.1 New Features (ENH)

- Simplify metadata input (Document Information Dictionary) (#1851)
- Extend cmap compatibilty to GBK_EUC_H/V (#1812)

48.22.2 Bug Fixes (BUG)

- Prevent infinite loop when no character follows after a comment (#1828)
- get_contents does not return ContentStream (#1847)
- Accept XYZ destination with zoom missing (default to zoom=0.0) (#1844)
- Cope with 1 Bit images (#1815)

48.22.3 Robustness (ROB)

- Handle missing /Type entry in Page tree (#1845)

48.22.4 Documentation (DOC)

- Expand file size explanations (#1835)
- Add comparison with pdfplumber (#1837)
- Clarify that PyPDF2 is dead (#1827)
- Add Hunter King as Contributor for #1806

48.22.5 Maintenance (MAINT)

- Refactor internal Encryption class (#1821)
- Add R parameter to generate_values (#1820)
- Make encryption_key parameter of write_to_stream optional (#1819)
- Prepare for adding AES enryption support (#1818)

Full Changelog

48.23 Version 3.8.1, 2023-04-23

48.23.1 Bug Fixes (BUG)

- Convert color space before saving (#1802)

48.23.2 Documentation (DOC)

- PDF/A (#1807)
- Use append instead of add_page
- Document core mechanics of pypdf (#1783)

Full Changelog

48.24 Version 3.8.0, 2023-04-16

48.24.1 New Features (ENH)

- Add transform method to Transformation class (#1765)
- Cope with UC2 fonts in text_extraction (#1785)

48.24.2 Robustness (ROB)

- Invalid startxref pointing 1 char before (#1784)

48.24.3 Maintenance (MAINT)

- Mark code handling old parameters as deprecated (#1798)

Full Changelog

48.25 Version 3.7.1, 2023-04-09

48.25.1 Security (SEC)

- Warn about PDF encryption security (#1755)

48.25.2 Robustness (ROB)

- Prevent loop in Cloning (#1770)
- Capture UnicodeDecodeError at PdfReader.pdf_header (#1768)

48.25.3 Documentation (DOC)

- Add .readthedocs.yaml and bump docs dependencies using `tox -e deps` (#1750, #1752)

48.25.4 Developer Experience (DEV)

- Make make_changelog.py idempotent

48.25.5 Maintenance (MAINT)

- Move generation of file identifiers to a method (#1760)

48.25.6 Testing (TST)

- Add xmp test (#1775)

Full Changelog

48.26 Version 3.7.0, 2023-03-26

48.26.1 Security (SEC)

- Use Python's secrets module instead of random module (#1748)

48.26.2 New Features (ENH)

- Add AnnotationBuilder.highlight text markup annotation (#1740)
- Add AnnotationBuilder.popup (#1665)
- Add AnnotationBuilder.polyline annotation support (#1726)
- Add clone_from parameter in PdfWriter constructor (#1703)

48.26.3 Bug Fixes (BUG)

- 'DictionaryObject' object has no attribute 'indirect_reference' (#1729)

48.26.4 Robustness (ROB)

- Handle params NullObject in decode_stream_data (#1738)

48.26.5 Documentation (DOC)

- Project scope (#1743)

48.26.6 Maintenance (MAINT)

- Add AnnotationFlag (#1746)
- Add LazyDict.**str** (#1727)

Full Changelog

48.27 Version 3.6.0, 2023-03-18

48.27.1 New Features (ENH)

- Extend PdfWriter.append() to PageObjects (#1704)
- Support qualified names in update_page_form_field_values (#1695)

48.27.2 Robustness (ROB)

- Tolerate streams without length field (#1717)
- Accept DictionaryObject in /D of NamedDestination (#1720)
- Widths def in cmap calls IndirectObject (#1719)

Full Changelog

48.28 Version 3.5.2, 2023-03-12

We discovered that compress_content_stream has to be applied to a page of the PdfWriter. It may not be applied to a page of the PdfReader!

48.28.1 Bug Fixes (BUG)

- compress_content_stream not readable in Adobe Acrobat (#1698)
- Pass logging parameters correctly in set_need_appearances_writer (#1697)
- Write /Root/AcroForm in set_need_appearances_writer (#1639)

48.28.2 Robustness (ROB)

- Allow more whitespaces within linearized file (#1701)

Full Changelog

48.29 Version 3.5.1, 2023-03-05

48.29.1 Robustness (ROB)

- Some attributes not copied in DictionaryObject._clone (#1635)
- Allow merging multiple time pages with annots (#1624)

48.29.2 Testing (TST)

- Replace pytest.mark.external by enable_socket (#1657)

Full Changelog

48.30 Version 3.5.0, 2023-02-26

48.30.1 New Features (ENH)

- Add reader.attachments public interface (#1611, #1661)
- Add PdfWriter.remove_objects_from_page(page: PageObject, to_delete: ObjectDeletionFlag) (#1648)
- Allow free-text annotation to have transparent border/background (#1664)

48.30.2 Bug Fixes (BUG)

- Allow decryption with empty password for AlgV5 (#1663)
- Let PdfWriter.pages return PageObject after calling `clone_document_from_reader()` (#1613)
- Invalid font pointed during merge_resources (#1641)

48.30.3 Robustness (ROB)

- Cope with invalid objects in IndirectObject.clone (#1637)
- Improve tolerance to invalid Names/Dests (#1658)
- Decode encoded values in get_fields (#1636)
- Let PdfWriter.merge cope with missing "/Fields" (#1628)

Full Changelog

48.31 Version 3.4.1, 2023-02-12

48.31.1 Bug Fixes (BUG)

- Switch from trimbox to cropbox when merging pages (#1622)
- Text extraction not working with one glyph to char sequence (#1620)

48.31.2 Robustness (ROB)

- Fix 2 cases of "object has no attribute 'indirect_reference'" (#1616)

48.31.3 Testing (TST)

- Add multiple retry on get_url for external PDF downloads (#1626)

Full Changelog

48.32 Version 3.4.0, 2023-02-05

NOTICE: pypdf changed the way it represents numbers parsed from PDF files. pypdf<3.4.0 represented numbers as Decimal, pypdf>=3.4.0 represents them as floats. Several other PDF libraries to this, as well as many PDF viewers. We hope to fix issues with too high precision like this and get a speed boost. In case your PDF documents rely on more than 18 decimals of precision you should check if it still works as expected. To clarify: This does not affect the text shown in PDF documents. It affects numbers, e.g. when graphics are drawn on the PDF or very exact positions are used. Typically, 5 decimals should be enough.

48.32.1 New Features (ENH)

- Enable merging forms with overlapping names (#1553)
- Add 'over' parameter to merge_transformend_page & co (#1567)

48.32.2 Bug Fixes (BUG)

- Fix getter of the PageObject.rotation property with an indirect object (#1602)
- Restore merge_transformed_page & co (#1567)
- Replace decimal by float (#1563)

48.32.3 Robustness (ROB)

- PdfWriter.remove_images: /Contents might not be in page_ref (#1598)

48.32.4 Developer Experience (DEV)

- Introduce ruff (#1586, #1609)

48.32.5 Maintenance (MAINT)

- Remove decimal (#1608)

Full Changelog

48.33 Version 3.3.0, 2023-01-22

48.33.1 New Features (ENH)

- Add page label support to PdfWriter (#1558)
- Accept inline images with space before EI (#1552)
- Add circle annotation support (#1556)
- Add polygon annotation support (#1557)
- Make merging pages produce a deterministic PDF (#1542, #1543)

48.33.2 Bug Fixes (BUG)

- Fix error in cmap extraction (#1544)
- Remove erroneous assertion check (#1564)
- Fix dictionary access of optional page label keys (#1562)

48.33.3 Robustness (ROB)

- Set ignore_eof=True for read_until_regex (#1521)

48.33.4 Documentation (DOC)

- Paper size (#1550)

48.33.5 Developer Experience (DEV)

- Fix broken combination of dependencies of docs.txt
- Annotate tests appropriately (#1551)

Full Changelog

48.34 Version 3.2.1, 2023-01-08

48.34.1 Bug Fixes (BUG)

- Accept hierarchical fields (#1529)

48.34.2 Documentation (DOC)

- Use google style docstrings (#1534)
- Fix linked markdown documents (#1537)

48.34.3 Developer Experience (DEV)

- Update docs config (#1535)

48.35 Version 3.2.0, 2022-12-31

48.35.1 Performance Improvement (PI)

- Help the specializing adpative interpreter (#1522)

48.35.2 New Features (ENH)

- Add support for page labels (#1519)

48.35.3 Bug Fixes (BUG)

- upgrade clone_document_root (#1520)

Full Changelog

48.36 Version 3.1.0, 2022-12-23

Move PyPDF2 to pypdf (#1513). This now it's all lowercase, no number in the name. For installation and for import. PyPDF2 will no longer receive updates. The community should move back to its roots.

If you were still using pyPdf or PyPDF2 < 2.0.0, I recommend reading the migration guide: https://pypdf.readthedocs.io/en/latest/user/migration-1-to-2.html

pypdf==3.1.0 is only different from PyPDF2==3.0.0 in the package name. Replacing "PyPDF2" by "pypdf" should be enough if you migrate from PyPDF2==`3.0.0` to pypdf==`3.1.0`.

Full Changelog

48.37 Version 3.0.0, 2022-12-22

48.37.1 BREAKING CHANGES

- Deprecate features with PyPDF2==3.0.0 (#1489)
- Refactor Fit / Zoom parameters (#1437)

48.37.2 New Features (ENH)

- Add Cloning (#1371)
- Allow int for indirect_reference in PdfWriter.get_object (#1490)

48.37.3 Documentation (DOC)

- How to read PDFs from S3 (#1509)
- Make MyST parse all links as simple hyperlinks (#1506)
- Changed 'latest' for 'stable' generated docs (#1495)
- Adjust deprecation procedure (#1487)

48.37.4 Maintenance (MAINT)

- Use typing.IO for file streams (#1498)

Full Changelog

48.38 Version 2.12.1, 2022-12-10

48.38.1 Documentation (DOC)

- Deduplicate extract_text docstring (#1485)
- How to cite PyPDF2 (#1476)

48.38.2 Maintenance (MAINT)

Consistency changes:

- indirect_ref/ido indirect_reference, dest page_destination (#1467)
- owner_pwd/user_pwd owner_password/user_password (#1483)
- position page_number in Merger.merge (#1482)
- indirect_ref indirect_reference (#1484)

Full Changelog

48.39 Version 2.12.0, 2022-12-10

48.39.1 New Features (ENH)

- Add support to extract gray scale images (#1460)
- Add 'threads' property to PdfWriter (#1458)
- Add 'open_destination' property to PdfWriter (#1431)
- Make PdfReader.get_object accept integer arguments (#1459)

48.39.2 Bug Fixes (BUG)

- Scale PDF annotations (#1479)

48.39.3 Robustness (ROB)

- Padding issue with AES encryption (#1469)
- Accept empty object as null objects (#1477)

48.39.4 Documentation (DOC)

- Add module documentation the PaperSize class (#1447)

48.39.5 Maintenance (MAINT)

- Use 'page_number' instead of 'pagenum' (#1365)
- Add List of pages to PageRangeSpec (#1456)

48.39.6 Testing (TST)

- Cleanup temporary files (#1454)
- Mark test_tounicode_is_identity as external (#1449)
- Use Ubuntu 20.04 for running CI test suite (#1452)

Full Changelog

48.40 Version 2.11.2, 2022-11-20

48.40.1 New Features (ENH)

- Add remove_from_tree (#1432)
- Add AnnotationBuilder.rectangle (#1388)

48.40.2 Bug Fixes (BUG)

- JavaScript executed twice (#1439)
- ToUnicode stores /Identity-H instead of stream (#1433)
- Declare Pillow as optional dependency (#1392)

48.40.3 Developer Experience (DEV)

- Link 'Full Changelog' automatically
- Modify read_string_from_stream to a benchmark (#1415)
- Improve error reporting of read_object (#1412)
- Test Python 3.11 (#1404)
- Extend Flake8 ignore list (#1410)
- Use correct pytest markers (#1407)
- Move project configuration to pyproject.toml (#1382)

Full Changelog

48.41 Version 2.11.1, 2022-10-09

48.41.1 Bug Fixes (BUG)

- td matrix (#1373)
- Cope with cmap from #1322 (#1372)

48.41.2 Robustness (ROB)

- Cope with str returned from get_data in cmap (#1380)

Full Changelog

48.42 Version 2.11.0, 2022-09-25

48.42.1 New Features (ENH)

- Addition of optional visitor-functions in extract_text() (#1252)
- Add metadata.creation_date and modification_date (#1364)
- Add PageObject.images attribute (#1330)

48.42.2 Bug Fixes (BUG)

- Lookup index in _xobj_to_image can be ByteStringObject (#1366)
- 'IndexError: index out of range' when using extract_text (#1361)
- Errors in transfer_rotation_to_content() (#1356)

48.42.3 Robustness (ROB)

- Ensure update_page_form_field_values does not fail if no fields (#1346)

Full Changelog

48.43 Version 2.10.9, 2022-09-18

48.43.1 New Features (ENH)

- Add rotation property and transfer_rotate_to_content (#1348)

48.43.2 Performance Improvements (PI)

- Avoid string concatenation with large embedded base64-encoded images (#1350)

48.43.3 Bug Fixes (BUG)

- Format floats using their intrinsic decimal precision (#1267)

48.43.4 Robustness (ROB)

- Fix merge_page for pages without resources (#1349)

Full Changelog

48.44 Version 2.10.8, 2022-09-14

48.44.1 New Features (ENH)

- Add PageObject.user_unit property (#1336)

48.44.2 Robustness (ROB)

- Improve NameObject reading/writing (#1345)

Full Changelog

48.45 Version 2.10.7, 2022-09-11

48.45.1 Bug Fixes (BUG)

- Fix Error in transformations (#1341)
- Decode #23 in NameObject (#1342)

48.45.2 Testing (TST)

- Use pytest.warns() for warnings, and .raises() for exceptions (#1325)

Full Changelog

48.46 Version 2.10.6, 2022-09-09

48.46.1 Robustness (ROB)

- Fix infinite loop due to Invalid object (#1331)
- Fix image extraction issue with superfluous whitespaces (#1327)

Full Changelog

48.47 Version 2.10.5, 2022-09-04

48.47.1 New Features (ENH)

- Process XRefStm (#1297)
- Auto-detect RTL for text extraction (#1309)

48.47.2 Bug Fixes (BUG)

- Avoid scaling cropbox twice (#1314)

48.47.3 Robustness (ROB)

- Fix offset correction in revised PDF (#1318)
- Crop data of /U and /O in encryption dictionary to 48 bytes (#1317)
- MultiLine bfrange in cmap (#1299)
- Cope with 2 digit codes in bfchar (#1310)
- Accept '/annn' charset as ASCII code (#1316)
- Log errors during Float / NumberObject initialization (#1315)
- Cope with corrupted entries in xref table (#1300)

48.47.4 Documentation (DOC)

- Migration guide (PyPDF2 1.x 2.x) (#1324)
- Creating a coverage report (#1319)
- Fix AnnotationBuilder.free_text example (#1311)
- Fix usage of page.scale by replacing it with page.scale_by (#1313)

48.47.5 Maintenance (MAINT)

- PdfReaderProtocol (#1303)
- Throw PdfReadError if Trailer can't be read (#1298)
- Remove catching OverflowException (#1302)

Full Changelog

48.48 Version 2.10.4, 2022-08-28

48.48.1 Robustness (ROB)

- Fix errors/warnings on no /Resources within extract_text (#1276)
- Add required line separators in ContentStream ArrayObjects (#1281)

48.48.2 Maintenance (MAINT)

- Use NameObject idempotency (#1290)

48.48.3 Testing (TST)

- Rectangle deletion (#1289)
- Add workflow tests (#1287)
- Remove files after tests ran (#1286)

48.48.4 Packaging (PKG)

- Add minimum version for typing_extensions requirement (#1277)

Full Changelog

48.49 Version 2.10.3, 2022-08-21

48.49.1 Robustness (ROB)

- Decrypt returns empty bytestring (#1258)

48.49.2 Developer Experience (DEV)

- Modify CI to better verify built package contents (#1244)

48.49.3 Maintenance (MAINT)

- Remove 'mine' as PdfMerger always creates the stream (#1261)
- Let PdfMerger._create_stream raise NotImplemented (#1251)
- password param of _security._alg32(. . .) is only a string, not bytes (#1259)
- Remove unreachable code in read_block_backwards (#1250) and sign function in _extract_text (#1262)

48.49.4 Testing (TST)

- Delete annotations (#1263)
- Close PdfMerger in tests (#1260)
- PdfReader.xmp_metadata workflow (#1257)
- Various PdfWriter (Layout, Bookmark deprecation) (#1249)

Full Changelog

48.50 Version 2.10.2, 2022-08-15

BUG: Add PyPDF2.generic to PyPI distribution

48.51 Version 2.10.1, 2022-08-15

48.51.1 Bug Fixes (BUG)

- TreeObject.remove_child had a non-PdfObject assignment for Count (#1233, #1234)
- Fix stream truncated prematurely (#1223)

48.51.2 Documentation (DOC)

- Fix docstring formatting (#1228)

48.51.3 Maintenance (MAINT)

- Split generic.py (#1229)

48.51.4 Testing (TST)

- Decrypt AlgV4 with owner password (#1239)

- AlgV5.generate_values (#1238)

- TreeObject.remove_child / empty_tree (#1235, #1236)

- create_string_object (#1232)

- Free-Text annotations (#1231)

- generic._base (#1230)

- Strict get fonts (#1226)

- Increase PdfReader coverage (#1219, #1225)

- Increase PdfWriter coverage (#1237)

- 100% coverage for utils.py (#1217)

- PdfWriter exception non-binary stream (#1218)

- Don't check coverage for deprecated code (#1216)

Full Changelog

48.52 Version 2.10.0, 2022-08-07

48.52.1 New Features (ENH)

- "with" support for PdfMerger and PdfWriter (#1193)

- Add AnnotationBuilder.text(...) to build text annotations (#1202)

48.52.2 Bug Fixes (BUG)

- Allow IndirectObjects as stream filters (#1211)

48.52.3 Documentation (DOC)

- Font scrambling

- Page vs Content scaling (#1208)

- Example for orientation parameter of extract_text (#1206)

- Fix AnnotationBuilder parameter formatting (#1204)

48.52.4 Developer Experience (DEV)

- Add flake8-print (#1203)

48.52.5 Maintenance (MAINT)

- Introduce WrongPasswordError / FileNotDecryptedError / EmptyFileError (#1201)

Full Changelog

48.53 Version 2.9.0, 2022-07-31

48.53.1 New Features (ENH)

- Add ability to add hex encoded colors to outline items (#1186)

- Add support for pathlib.Path in PdfMerger.merge (#1190)

- Add link annotation (#1189)

- Add capability to filter text extraction by orientation (#1175)

48.53.2 Bug Fixes (BUG)

- Named Dest in PDF1.1 (#1174)

- Incomplete Graphic State save/restore (#1172)

48.53.3 Documentation (DOC)

- Update changelog url in package metadata (#1180)

- Mantion camelot for table extraction (#1179)

- Mention pyHanko for signing PDF documents (#1178)

- Weow have CMAP support since a while (#1177)

48.53.4 Maintenance (MAINT)

- Consistant usage of warnings / log messages (#1164)
- Consistent terminology for outline items (#1156)

Full Changelog

48.54 Version 2.8.1, 2022-07-25

48.54.1 Bug Fixes (BUG)

- u_hash in AlgV4.compute_key (#1170)

48.54.2 Robustness (ROB)

- Fix loading of file from #134 (#1167)
- Cope with empty DecodeParams (#1165)

48.54.3 Documentation (DOC)

- Typo in merger deprecation warning message (#1166)

48.54.4 Maintenance (MAINT)

- Package updates; solve mypy strict remarks (#1163)

48.54.5 Testing (TST)

- Add test from #325 (#1169)

Full Changelog

48.55 Version 2.8.0, 2022-07-24

48.55.1 New Features (ENH)

- Add writer.add_annotation, page.annotations, and generic.AnnotationBuilder (#1120)

48.55.2 Bug Fixes (BUG)

- Set /AS for /Btn form fields in writer (#1161)
- Ignore if /Perms verify failed (#1157)

48.55.3 Robustness (ROB)

- Cope with utf16 character for space calculation (#1155)
- Cope with null params for FitH / FitV destination (#1152)
- Handle outlines without valid destination (#1076)

48.55.4 Developer Experience (DEV)

- Introduce _utils.logger_warning (#1148)

48.55.5 Maintenance (MAINT)

- Break up parse_to_unicode (#1162)
- Add diagnostic output to exception in read_from_stream (#1159)
- Reduce PdfReader.read complexity (#1151)

48.55.6 Testing (TST)

- Add workflow tests found by arc testing (#1154)
- Decrypt file which is not encrypted (#1149)
- Test CryptRC4 encryption class; test image extraction filters (#1147)

Full Changelog

48.56 Version 2.7.0, 2022-07-21

48.56.1 New Features (ENH)

- Add `outline_count` property (#1129)

48.56.2 Bug Fixes (BUG)

- Make reader.get_fields also return dropdowns with options (#1114)
- Add deprecated EncodedStreamObject functions back until PyPDF2==3.0.0 (#1139)

48.56.3 Robustness (ROB)

- Cope with missing /W entry (#1136)
- Cope with invalid parent xref (#1133)

48.56.4 Documentation (DOC)

- Contributors file (#1132)
- Fix type in signature of PdfWriter.add_uri (#1131)

48.56.5 Developer Experience (DEV)

- Add .git-blame-ignore-revs (#1141)

48.56.6 Code Style (STY)

- Fixing typos (#1137)
- Re-use code via get_outlines_property in tests (#1130)

Full Changelog

48.57 Version 2.6.0, 2022-07-17

48.57.1 New Features (ENH)

- Add color and font_format to PdfReader.outlines[i] (#1104)
- Extract Text Enhancement (whitespaces) (#1084)

48.57.2 Bug Fixes (BUG)

- Use `build_destination` for named destination outlines (#1128)
- Avoid a crash when a ToUnicode CMap has an empty dstString in beginbfchar (#1118)
- Prevent deduplication of PageObject (#1105)
- None-check in DictionaryObject.read_from_stream (#1113)
- Avoid IndexError in _cmap.parse_to_unicode (#1110)

48.57.3 Documentation (DOC)

- Explanation for git submodule
- Watermark and stamp (#1095)

48.57.4 Maintenance (MAINT)

- Text extraction improvements (#1126)
- Destination.color returns ArrayObject instead of tuple as fallback (#1119)
- Use add_bookmark_destination in add_bookmark (#1100)
- Use add_bookmark_destination in add_bookmark_dict (#1099)

48.57.5 Testing (TST)

- Add test for arab text (#1127)
- Add xfail for decryption fail (#1125)
- Add xfail test for IndexError when extracting text (#1124)
- Add MCVE showing outline title issue (#1123)

48.57.6 Code Style (STY)

- Use IntFlag for permissions_flag / update_page_form_field_values (#1094)
- Simplify code (#1101)

Full Changelog

48.58 Version 2.5.0, 2022-07-10

48.58.1 New Features (ENH)

- Add support for indexed color spaces / BitsPerComponent for decoding PNGs (#1067)
- Add PageObject._get_fonts (#1083)

48.58.2 Performance Improvements (PI)

- Use iterative DFS in PdfWriter._sweep_indirect_references (#1072)

48.58.3 Bug Fixes (BUG)

- Let Page.scale also scale the crop-/trim-/bleed-/artbox (#1066)
- Column default for CCITTFaxDecode (#1079)

48.58.4 Robustness (ROB)

- Guard against None-value in _get_outlines (#1060)

48.58.5 Documentation (DOC)

- Stamps and watermarks (#1082)
- OCR vs PDF text extraction (#1081)
- Python Version support
- Formatting of CHANGELOG

48.58.6 Developer Experience (DEV)

- Cache downloaded files (#1070)
- Speed-up for CI (#1069)

48.58.7 Maintenance (MAINT)

- Set page.rotate(angle: int) (#1092)
- Issue #416 was fixed by #1015 (#1078)

48.58.8 Testing (TST)

- Image extraction (#1080)
- Image extraction (#1077)

48.58.9 Code Style (STY)

- Apply black
- Typo in Changelog

Full Changelog

48.59 Version 2.4.2, 2022-07-05

48.59.1 New Features (ENH)

- Add PdfReader.xfa attribute (#1026)

48.59.2 Bug Fixes (BUG)

- Wrong page inserted when PdfMerger.merge is done (#1063)
- Resolve IndirectObject when it refers to a free entry (#1054)

48.59.3 Developer Experience (DEV)

- Added {posargs} to tox.ini (#1055)

48.59.4 Maintenance (MAINT)

- Remove PyPDF2._utils.bytes_type (#1053)

48.59.5 Testing (TST)

- Scale page (indirect rect object) (#1057)
- Simplify pathlib PdfReader test (#1056)
- IndexError of VirtualList (#1052)
- Invalid XML in xmp information (#1051)
- No pycryptodome (#1050)
- Increase test coverage (#1045)

48.59.6 Code Style (STY)

- DOC of compress_content_streams (#1061)
- Minimize diff for #879 (#1049)

Full Changelog

48.60 Version 2.4.1, 2022-06-30

48.60.1 New Features (ENH)

- Add writer.pdf_header property (getter and setter) (#1038)

48.60.2 Performance Improvements (PI)

- Remove b_ call in FloatObject.write_to_stream (#1044)
- Check duplicate objects in writer._sweep_indirect_references (#207)

48.60.3 Documentation (DOC)

- How to surppress exceptions/warnings/log messages (#1037)
- Remove hyphen from lossless (#1041)
- Compression of content streams (#1040)
- Fix inconsistent variable names in add-watermark.md (#1039)
- File size reduction
- Add CHANGELOG to the rendered docs (#1023)

48.60.4 Maintenance (MAINT)

- Handle XML error when reading XmpInformation (#1030)
- Deduplicate Code / add mutmut config (#1022)

48.60.5 Code Style (STY)

- Use unnecessary one-line function / class attribute (#1043)
- Docstring formatting (#1033)

Full Changelog

48.61 Version 2.4.0, 2022-06-26

48.61.1 New Features (ENH):

- Support R6 decrypting (#1015)
- Add PdfReader.pdf_header (#1013)

48.61.2 Performance Improvements (PI):

- Remove ord_ calls (#1014)

48.61.3 Bug Fixes (BUG):

- Fix missing page for bookmark (#1016)

48.61.4 Robustness (ROB):

- Deal with invalid Destinations (#1028)

48.61.5 Documentation (DOC):

- get_form_text_fields does not extract dropdown data (#1029)
- Adjust PdfWriter.add_uri docstring
- Mention crypto extra_requires for installation (#1017)

48.61.6 Developer Experience (DEV):

- Use /n line endings everywhere (#1027)
- Adjust string formatting to be able to use mutmut (#1020)
- Update Bug report template

Full Changelog

48.62 Version 2.3.1, 2022-06-19

BUG: Forgot to add the interal `_codecs` subpackage.

Full Changelog

48.63 Version 2.3.0, 2022-06-19

The highlight of this release is improved support for file encryption (AES-128 and AES-256, R5 only). See #749 for the amazing work of @exiledkingcc Thank you

48.63.1 Deprecations (DEP)

- Rename names to be PEP8-compliant (#967)
- `PdfWriter.get_page`: the pageNumber parameter is renamed to page_number
- `PyPDF2.filters`:
  - For all classes, a parameter rename: decodeParms decode_parms
  - decodeStreamData decode_stream_data
- `PyPDF2.xmp`:
  - XmpInformation.rdfRoot XmpInformation.rdf_root
  - XmpInformation.xmp_createDate XmpInformation.xmp_create_date

– XmpInformation.xmp_creatorTool XmpInformation.xmp_creator_tool

– XmpInformation.xmp_metadataDate XmpInformation.xmp_metadata_date

– XmpInformation.xmp_modifyDate XmpInformation.xmp_modify_date

– XmpInformation.xmpMetadata XmpInformation.xmp_metadata

– XmpInformation.xmpmm_documentId XmpInformation.xmpmm_document_id

– XmpInformation.xmpmm_instanceId XmpInformation.xmpmm_instance_id

- `PyPDF2.generic`:

– readHexStringFromStream read_hex_string_from_stream

– initializeFromDictionary initialize_from_dictionary

– createStringObject create_string_object

– TreeObject.hasChildren TreeObject.has_children

– TreeObject.emptyTree TreeObject.empty_tree

48.63.2 New Features (ENH)

- Add decrypt support for V5 and AES-128, AES-256 (R5 only) (#749)

48.63.3 Robustness (ROB)

- Fix corrupted (wrongly) linear PDF (#1008)

48.63.4 Maintenance (MAINT)

- Move PDF_Samples folder into ressources
- Fix typos (#1007)

48.63.5 Testing (TST)

- Improve encryption/decryption test (#1009)
- Add merger test cases with real PDFs (#1006)
- Add mutmut config

48.63.6 Code Style (STY)

- Put pure data mappings in separate files (#1005)
- Make encryption module private, apply pre-commit (#1010)

Full Changelog

48.64 Version 2.2.1, 2022-06-17

48.64.1 Performance Improvements (PI)

- Remove b_ calls (#992, #986)
- Apply improvements to _utils suggested by perflint (#993)

48.64.2 Robustness (ROB)

- utf-16-be codec can't decode (…) (#995)

48.64.3 Documentation (DOC)

- Remove reference to Scripts (#987)

48.64.4 Developer Experience (DEV)

- Fix type annotations for add_bookmarks (#1000)

48.64.5 Testing (TST)

- Add test for PdfMerger (#1001)
- Add tests for XMP information (#996)
- reader.get_fields / zlib issue / LZW decode issue (#1004)
- reader.get_fields with report generation (#1002)
- Improve test coverage by extracting texts (#998)

48.64.6 Code Style (STY)

- Apply fixes suggested by pylint (#999)

Full Changelog

48.65 Version 2.2.0, 2022-06-13

The 2.2.0 release improves text extraction again via (#969):

- Improvements around /Encoding / /ToUnicode
- Extraction of CMaps improved
- Fallback for font def missing
- Support for /Identity-H and /Identity-V: utf-16-be
- Support for /GB-EUC-H / /GB-EUC-V / GBp/c-EUC-H / /GBpc-EUC-V (beta release for evaluation)
- Arabic (for evaluation)

- Whitespace extraction improvements

Those changes should mainly improve the text extraction for non-ASCII alphabets, e.g. Russian / Chinese / Japanese / Korean / Arabic.

Full Changelog

48.66 Version 2.1.1, 2022-06-12

48.66.1 New Features (ENH)

- Add support for pathlib as input for PdfReader (#979)

48.66.2 Performance Improvements (PI)

- Optimize read_next_end_line (#646)

48.66.3 Bug Fixes (BUG)

- Adobe Acrobat 'Would you like to save this file?' (#970)

48.66.4 Documentation (DOC)

- Notes on annotations (#982)
- Who uses PyPDF2
- intendet \xe2\x9e\x94 in robustness page (#958)

48.66.5 Maintenance (MAINT)

- pre-commit / requirements.txt updates (#977)
- Mark read_next_end_line as deprecated (#965)
- Export `PageObject` in PyPDF2 root (#960)

48.66.6 Testing (TST)

- Add MCVE of issue #416 (#980)
- FlateDecode.decode decodeParms (#964)
- Xmp module (#962)
- utils.paeth_predictor (#959)

48.66.7 Code Style (STY)

- Use more tuples and list/dict comprehensions (#976)

Full Changelog

48.67 Version 2.1.0, 2022-06-06

The highlight of the 2.1.0 release is the most massive improvement to the text extraction capabilities of PyPDF2 since 2016 A very big thank you goes to pubpub-zz who took a lot of time and knowledge about the PDF format to finally get those improvements into PyPDF2. Thank you

In case the new function causes any issues, you can use `_extract_text_old` for the old functionality. Please also open a bug ticket in that case.

There were several people who have attempted to bring similar improvements to PyPDF2. All of those were valuable. The main reason why they didn't get merged is the big amount of open PRs / issues. pubpub-zz was the most comprehensive PR which also incorporated the latest changes of PyPDF2 2.0.0.

Thank you to VictorCarlquist for #858 and asabramo for #464

48.67.1 New Features (ENH)

- Massive text extraction improvement (#924). Closed many open issues:
    - Exceptions / missing spaces in extract_text() method (#17)
        * Whitespace issues in extract_text() (#42)
        * pypdf2 reads the hifenated words in a new line (#246)
    - PyPDF2 failing to read unicode character (#37)
        * Unable to read bullets (#230)
    - ExtractText yields nothing for apparently good PDF (#168)
    - Encoding issue in extract_text() (#235)
    - extractText() doesn't work on Chinese PDF (#252)
    - encoding error (#260)
    - Trouble with apostophes in names in text "O'Doul" (#384)
    - extract_text works for some PDF files, but not the others (#437)
    - Euro sign not being recognized by extractText (#443)
    - Failed extracting text from French texts (#524)
    - extract_text doesn't extract ligatures correctly (#598)
    - reading spanish text - mark convert issue (#635)
    - Read PDF changed from text to random symbols (#654)
    - .extractText() reads / as 1. (#789)
- Update glyphlist (#947) - inspired by #464
- Allow adding PageRange objects (#948)

48.67.2 Bug Fixes (BUG)

- Delete .python-version file (#944)
- Compare StreamObject.decoded_self with None (#931)

48.67.3 Robustness (ROB)

- Fix some conversion errors on non conform PDF (#932)

48.67.4 Documentation (DOC)

- Elaborate on PDF text extraction difficulties (#939)
- Add logo (#942)
- rotate vs Transformation().rotate (#937)
- Example how to use PyPDF2 with AWS S3 (#938)
- How to deprecate (#930)
- Fix typos on robustness page (#935)
- Remove scripts (pdfcat) from docs (#934)

48.67.5 Developer Experience (DEV)

- Ignore .python-version file
- Mark deprecated code with no-cover (#943)
- Automatically create Github releases from tags (#870)

48.67.6 Testing (TST)

- Text extraction for non-latin alphabets (#954)
- Ignore PdfReadWarning in benchmark (#949)
- writer.remove_text (#946)
- Add test for Tree and _security (#945)

48.67.7 Code Style (STY)

- black, isort, Flake8, splitting buildCharMap (#950)

Full Changelog

48.68 Version 2.0.0, 2022-06-01

The 2.0.0 release of PyPDF2 includes three core changes:

1. Dropping support for Python 3.5 and older.

2. Introducing type annotations.

3. Interface changes, mostly to have PEP8-compliant names

We introduced a deprecation process that hopefully helps users to avoid unexpected breaking changes.

48.68.1 Breaking Changes (DEP)

- PyPDF2 2.0 requires Python 3.6+. Python 2.7 and 3.5 support were dropped.

- PdfFileReader: The "warndest" parameter was removed

- PdfFileReader and PdfFileMerger no longer have the `overwriteWarnings` parameter. The new behavior is `overwriteWarnings=False`.

- merger: OutlinesObject was removed without replacement.

- merger.py _merger.py: You must import PdfFileMerger from PyPDF2 directly.

- utils:

    - `ConvertFunctionsToVirtualList` was removed

    - `formatWarning` was removed

    - `isInt(obj)`: Use `instance(obj, int)` instead

    - `u_(s)`: Use `s` directly

    - `chr_(c)`: Use `chr(c)` instead

    - `barray(b)`: Use `bytearray(b)` instead

    - `isBytes(b)`: Use `instance(b, type(bytes()))` instead

    - `xrange_fn`: Use `range` instead

    - `string_type`: Use `str` instead

    - `isString(s)`: Use `instance(s, str)` instead

    - `_basestring`: Use `str` instead

    - All Exceptions are now in `PyPDF2.errors`:

        * PageSizeNotDefinedError

        * PdfReadError

        * PdfReadWarning

        * PyPdfError

- `PyPDF2.pdf` (the `pdf` module) no longer exists. The contents were moved with the library. You should most likely import directly from PyPDF2 instead. The `RectangleObject` is in `PyPDF2.generic`.

- The `Resources`, `Scripts`, and `Tests` will no longer be part of the distribution files on PyPI. This should have little to no impact on most people. The `Tests` are renamed to `tests`, the `Resources` are renamed to `resources`. Both are still in the git repository. The `Scripts` are now in cpdf. `Sample_Code` was moved to the `docs`.

For a full list of deprecated functions, please see the changelog of version 1.28.0.

48.68.2 New Features (ENH)

- Improve space setting for text extraction (#922)
- Allow setting the decryption password in `PdfReader.__init__` (#920)
- Add Page.add_transformation (#883)

48.68.3 Bug Fixes (BUG)

- Fix error adding transformation to page without /Contents (#908)

48.68.4 Robustness (ROB)

- Cope with invalid length in streams (#861)

48.68.5 Documentation (DOC)

- Fix style of 1.25 and 1.27 patch notes (#927)
- Transformation (#907)

48.68.6 Developer Experience (DEV)

- Create flake8 config file (#916)
- Use relative imports (#875)

48.68.7 Maintenance (MAINT)

- Use Python 3.6 language features (#849)
- Add wrapper function for PendingDeprecationWarnings (#928)
- Use new PEP8 compliant names (#884)
- Explicitly represent transformation matrix (#878)
- Inline PAGE_RANGE_HELP string (#874)
- Remove unnecessary generics imports (#873)
- Remove star imports (#865)
- merger.py _merger.py (#864)
- Type annotations for all functions/methods (#854)
- Add initial type support with mypy (#853)

48.68.8 Testing (TST)

- Regression test for xmp_metadata converter (#923)
- Checkout submodule sample-files for benchmark
- Add text extracting performance benchmark
- Use new PyPDF2 API in benchmark (#902)
- Make test suite fail for uncaught warnings (#892)
- Remove -OO testrun from CI (#901)
- Improve tests for convert_to_int (#899)

Full Changelog

48.69 PyPDF2 1.X

See *CHANGELOG PyPDF2 1.X*

CHANGELOG OF PYPDF2 1.X

49.1 Version 1.28.4, 2022-05-29

Bug Fixes (BUG):

- XmpInformation._converter_date was unusable (#921)

Full Changelog

49.2 Version 1.28.3, 2022-05-28

49.2.1 Deprecations (DEP)

- PEP8 renaming (#905)

49.2.2 Bug Fixes (BUG)

- XmpInformation missing method _getText (#917)
- Fix PendingDeprecationWarning on _merge_page (#904)

Full Changelog

49.3 Version 1.28.2, 2022-05-23

49.3.1 Bug Fixes (BUG)

- PendingDeprecationWarning for getContents (#893)
- PendingDeprecationWarning on using PdfMerger (#891)

Full Changelog

49.4 Version 1.28.1, 2022-05-22

49.4.1 Bug Fixes (BUG)

- Incorrectly show deprecation warnings on internal usage (#887)

49.4.2 Maintenance (MAINT)

- Add stacklevel=2 to deprecation warnings (#889)
- Remove duplicate warnings imports (#888)

Full Changelog

49.5 Version 1.28.0, 2022-05-22

This release adds a lot of deprecation warnings in preparation of the PyPDF2 2.0.0 release. The changes are mostly using snake_case function-, method-, and variable-names as well as using properties instead of getter-methods.

Maintenance (MAINT):

- Remove IronPython Fallback for zlib (#868)

Full Changelog

49.5.1 Deprecations (DEP)

- Make the `PyPDF2.utils` module private
- Rename of core classes:
  - PdfFileReader PdfReader
  - PdfFileWriter PdfWriter
  - PdfFileMerger PdfMerger
- Use PEP8 conventions for function names and parameters
- If a property and a getter-method are both present, use the property

Details

In many places:

- getObject get_object
- writeToStream write_to_stream
- readFromStream read_from_stream

PyPDF2.generic

- readObject read_object
- convertToInt convert_to_int

- DocumentInformation.getText DocumentInformation._get_text : This method should typically not be used; please let me know if you need it.

PdfReader class:

- `reader.getPage(pageNumber) reader.pages[page_number]`

- `reader.getNumPages() / reader.numPages len(reader.pages)`

- getDocumentInfo metadata

- flattenedPages attribute flattened_pages

- resolvedObjects attribute resolved_objects

- xrefIndex attribute xref_index

- getNamedDestinations / namedDestinations attribute named_destinations

- getPageLayout / pageLayout page_layout attribute

- getPageMode / pageMode page_mode attribute

- getIsEncrypted / isEncrypted is_encrypted attribute

- getOutlines get_outlines

- readObjectHeader read_object_header

- cacheGetIndirectObject cache_get_indirect_object

- cacheIndirectObject cache_indirect_object

- getDestinationPageNumber get_destination_page_number

- readNextEndLine read_next_end_line

- _zeroXref _zero_xref

- _authenticateUserPassword _authenticate_user_password

- _pageId2Num attribute _page_id2num

- _buildDestination _build_destination

- _buildOutline _build_outline

- _getPageNumberByIndirect(indirectRef) _get_page_number_by_indirect(indirect_ref)

- _getObjectFromStream _get_object_from_stream

- _decryptObject _decrypt_object

- _flatten(..., indirectRef) _flatten(..., indirect_ref)

- _buildField _build_field

- _checkKids _check_kids

- _writeField _write_field

- _write_field(..., fieldAttributes) _write_field(..., field_attributes)

- _read_xref_subsections(..., getEntry, ...) _read_xref_subsections(..., get_entry, ...)

PdfWriter class:

- `writer.getPage(pageNumber) writer.pages[page_number]`

- `writer.getNumPages() len(writer.pages)`

- addMetadata add_metadata

- addPage add_page

- addBlankPage add_blank_page

- addAttachment(fname, fdata) add_attachment(filename, data)

- insertPage insert_page

- insertBlankPage insert_blank_page

- appendPagesFromReader append_pages_from_reader

- updatePageFormFieldValues update_page_form_field_values

- cloneReaderDocumentRoot clone_reader_document_root

- cloneDocumentFromReader clone_document_from_reader

- getReference get_reference

- getOutlineRoot get_outline_root

- getNamedDestRoot get_named_dest_root

- addBookmarkDestination add_bookmark_destination

- addBookmarkDict add_bookmark_dict

- addBookmark add_bookmark

- addNamedDestinationObject add_named_destination_object

- addNamedDestination add_named_destination

- removeLinks remove_links

- removeImages(ignoreByteStringObject) remove_images(ignore_byte_string_object)

- removeText(ignoreByteStringObject) remove_text(ignore_byte_string_object)

- addURI add_uri

- addLink add_link

- getPage(pageNumber) get_page(page_number)

- getPageLayout / setPageLayout / pageLayout page_layout attribute

- getPageMode / setPageMode / pageMode page_mode attribute

- _addObject _add_object

- _addPage _add_page

- _sweepIndirectReferences _sweep_indirect_references

PdfMerger class

- __init__ parameter: strict=True strict=False (the PdfFileMerger still has the old default)

- addMetadata add_metadata

- addNamedDestination add_named_destination

- setPageLayout set_page_layout

- setPageMode set_page_mode

Page class:

- artBox / bleedBox/ cropBox/ mediaBox / trimBox artbox / bleedbox/ cropbox/ mediabox / trimbox

  - getWidth, getHeight width / height

  - getLowerLeft_x / getUpperLeft_x left

  - getUpperRight_x / getLowerRight_x right

  - getLowerLeft_y / getLowerRight_y bottom

  - getUpperRight_y / getUpperLeft_y top

  - getLowerLeft / setLowerLeft lower_left property

  - upperRight upper_right

- mergePage merge_page

- rotateClockwise / rotateCounterClockwise rotate_clockwise

- _mergeResources _merge_resources

- _contentStreamRename _content_stream_rename

- _pushPopGS _push_pop_gs

- _addTransformationMatrix _add_transformation_matrix

- _mergePage _merge_page

XmpInformation class:

- getElement(…, aboutUri, …) get_element(…, about_uri, …)

- getNodesInNamespace(…, aboutUri, …) get_nodes_in_namespace(…, aboutUri, …)

- _getText _get_text

utils.py:

- matrixMultiply matrix_multiply

- RC4_encrypt is moved to the security module

49.6 Version 1.27.12, 2022-05-02

49.6.1 Bug Fixes (BUG)

- _rebuild_xref_table expects trailer to be a dict (#857)

49.6.2 Documentation (DOC)

- Security Policy

Full Changelog

49.7 Version 1.27.11, 2022-05-02

49.7.1 Bug Fixes (BUG)

- Incorrectly issued xref warning/exception (#855)

Full Changelog

49.8 Version 1.27.10, 2022-05-01

49.8.1 Robustness (ROB)

- Handle missing destinations in reader (#840)
- warn-only in readStringFromStream (#837)
- Fix corruption in startxref or xref table (#788 and #830)

49.8.2 Documentation (DOC)

- Project Governance (#799)
- History of PyPDF2
- PDF feature/version support (#816)
- More details on text parsing issues (#815)

49.8.3 Developer Experience (DEV)

- Add benchmark command to Makefile
- Ignore IronPython parts for code coverage (#826)

49.8.4 Maintenance (MAINT)

- Split pdf module (#836)
- Separated CCITTFax param parsing/decoding (#841)
- Update requirements files

49.8.5 Testing (TST)

- Use external repository for larger/more PDFs for testing (#820)
- Swap incorrect test names (#838)
- Add test for PdfFileReader and page properties (#835)
- Add tests for PyPDF2.generic (#831)
- Add tests for utils, form fields, PageRange (#827)
- Add test for ASCII85Decode (#825)

- Add test for FlateDecode (#823)
- Add test for filters.ASCIIHexDecode (#822)

49.8.6 Code Style (STY)

- Apply pre-commit (black, isort) + use snake_case variables (#832)
- Remove debug code (#828)
- Documentation, Variable names (#839)

Full Changelog

49.9 Version 1.27.9, 2022-04-24

A change I would like to highlight is the performance improvement for large PDF files (#808)

49.9.1 New Features (ENH)

- Add papersizes (#800)
- Allow setting permission flags when encrypting (#803)
- Allow setting form field flags (#802)

49.9.2 Bug Fixes (BUG)

- TypeError in xmp._converter_date (#813)
- Improve spacing for text extraction (#806)
- Fix PDFDocEncoding Character Set (#809)

49.9.3 Robustness (ROB)

- Use null ID when encrypted but no ID given (#812)
- Handle recursion error (#804)

49.9.4 Documentation (DOC)

- CMaps (#811)
- The PDF Format + commit prefixes (#810)
- Add compression example (#792)

49.9.5 Developer Experience (DEV)

- Add Benchmark for Performance Testing (#781)

49.9.6 Maintenance (MAINT)

- Validate PDF magic byte in strict mode (#814)
- Make PdfFileMerger.addBookmark() behave life PdfFileWriters' (#339)
- Quadratic runtime while parsing reduced to linear (#808)

49.9.7 Testing (TST)

- Newlines in text extraction (#807)

Full Changelog

49.10 Version 1.27.8, 2022-04-21

49.10.1 Bug Fixes (BUG)

- Use 1MB as offset for readNextEndLine (#321)
- 'PdfFileWriter' object has no attribute 'stream' (#787)

49.10.2 Robustness (ROB)

- Invalid float object; use 0 as fallback (#782)

49.10.3 Documentation (DOC)

- Robustness (#785)

Full Changelog

49.11 Version 1.27.7, 2022-04-19

49.11.1 Bug Fixes (BUG)

- Import exceptions from PyPDF2.errors in PyPDF2.utils (#780)

49.11.2 Code Style (STY)

- Naming in 'make_changelog.py'

49.12 Version 1.27.6, 2022-04-18

49.12.1 Deprecations (DEP)

- Remove support for Python 2.6 and older (#776)

49.12.2 New Features (ENH)

- Extract document permissions (#320)

49.12.3 Bug Fixes (BUG)

- Clip by trimBox when merging pages, which would otherwise be ignored (#240)
- Add overwriteWarnings parameter PdfFileMerger (#243)
- IndexError for getPage() of decryped file (#359)
- Handle cases where decodeParms is an ArrayObject (#405)
- Updated PDF fields don't show up when page is written (#412)
- Set Linked Form Value (#414)
- Fix zlib -5 error for corrupt files (#603)
- Fix reading more than last1K for EOF (#642)
- Acciental import

49.12.4 Robustness (ROB)

- Allow extra whitespace before "obj" in readObjectHeader (#567)

49.12.5 Documentation (DOC)

- Link to pdftoc in Sample_Code (#628)
- Working with annotations (#764)
- Structure history

49.12.6 Developer Experience (DEV)

- Add issue templates (#765)
- Add tool to generate changelog

49.12.7 Maintenance (MAINT)

- Use grouped constants instead of string literals (#745)
- Add error module (#768)
- Use decorators for @staticmethod (#775)
- Split long functions (#777)

49.12.8 Testing (TST)

- Run tests in CI once with -OO Flags (#770)
- Filling out forms (#771)
- Add tests for Writer (#772)
- Error cases (#773)
- Check Error messages (#769)
- Regression test for issue #88
- Regression test for issue #327

49.12.9 Code Style (STY)

- Make variable naming more consistent in tests

Full changelog

49.13 Version 1.27.5, 2022-04-15

49.13.1 Security (SEC)

- ContentStream_readInlineImage had potential infinite loop (#740)

49.13.2 Bug fixes (BUG)

- Fix merging encrypted files (#757)
- CCITTFaxDecode decodeParms can be an ArrayObject (#756)

49.13.3 Robustness improvements (ROBUST)

• title sometimes None (#744)

49.13.4 Documentation (DOC)

• Adjust short description of the package

49.13.5 Tests and Test setup (TST)

• Rewrite JS tests from unittest to pytest (#746)

• Increase Test coverage, mainly with filters (#756)

• Add test for inline images (#758)

49.13.6 Developer Experience Improvements (DEV)

• Remove unused Travis-CI configuration (#747)

• Show code coverage (#754, #755)

• Add mutmut (#760)

49.13.7 Miscellaneous

• STY: Closing file handles, explicit exports, … (#743)

Full Changelog

49.14 Version 1.27.4, 2022-04-12

49.14.1 Bug fixes (BUG)

• Guard formatting of `__init__.__doc__` string (#738)

49.14.2 Packaging (PKG)

• Add more precise license field to setup (#733)

49.14.3 Testing (TST)

- Add test for issue #297

49.14.4 Miscellaneous

- DOC: Miscallenious Miscellaneous (Typo)
- TST: Fix CI triggering (master main) (#739)
- STY: Fix various style issues (#742)

Full Changelog

49.15 Version 1.27.3, 2022-04-10

- PKG: Make Tests not a subpackage (#728)
- BUG: Fix ASCII85Decode.decode assertion (#729)
- BUG: Error in Chinese character encoding (#463)
- BUG: Code duplication in Scripts/2-up.py
- ROBUST: Guard 'obj.writeToStream' with 'if obj is not None'
- ROBUST: Ignore a /Prev entry with the value 0 in the trailer
- MAINT: Remove Sample_Code (#726)
- TST: Close file handle in test_writer (#722)
- TST: Fix test_get_images (#730)
- DEV: Make tox use pytest and add more Python versions (#721)
- DOC: Many (#720, #723-725, #469)

Full Changelog

49.16 Version 1.27.2, 2022-04-09

- Add Scripts (including `pdfcat`), Resources, Tests, and Sample_Code back to PyPDF2. It was removed by accident in 1.27.0, but might get removed with 2.0.0 See discussions/718.

Full Changelog

49.17 Version 1.27.1, 2022-04-08

- Fixed project links on PyPI page after migration from mstamy2 to MartinThoma to the py-pdf organization on GitHub
- Documentation is now at pypdf2.readthedocs.io

Full Changelog

49.18 Version 1.27.0, 2022-04-07

Features:

- Add alpha channel support for png files in Script (#614)

49.18.1 Bug fixes (BUG)

- Fix formatWarning for filename without slash (#612)
- Add whitespace between words for extractText() (#569, #334)
- "invalid escape sequence" SyntaxError (#522)
- Avoid error when printing warning in pythonw (#486)
- Stream operations can be List or Dict (#665)

49.18.2 Documentation (DOC)

- Added Scripts/pdf-image-extractor.py
- Documentation improvements (#550, #538, #324, #426, #394)

49.18.3 Tests and Test setup (TST)

- Add Github Action which automatically run unit tests via pytest and static code analysis with Flake8 (#660)
- Add several unit tests (#661, #663)
- Add .coveragerc to create coverage reports

49.18.4 Developer Experience Improvements (DEV)

- Pre commit: Developers can now `pre-commit install` to avoid tiny issues like trailing whitespaces

49.18.5 Miscellaneous

- Add the LICENSE file to the distributed packages (#288)
- Use setuptools instead of distutils (#599)
- Improvements for the PyPI page (#644)
- Python 3 changes (#504, #366)

Full Changelog

49.19 Version 1.26.0, 2016-05-18

- NOTE: Active maintenance on PyPDF2 is resuming after a hiatus
- Fixed a bug where image resources where incorrectly overwritten when merging pages
- Added dictionary for JavaScript actions to the root (louib)
- Added unit tests for the JS functionality (louib)
- Add more Python 3 compatibility when reading inline images (im2703 and (VyacheslavHashov)
- Return NullObject instead of raising error when failing to resolve object (ctate)
- Don't output warning for non-zeroed xref table when strict=False (BenRussert)
- Remove extraneous zeroes from output formatting (speedplane)
- Fix bug where reading an inline image would cut off prematurely in certain cases (speedplane)

49.20 Version 1.25.1, 2015-07-20

- Fix bug when parsing inline images. Occurred when merging certain pages with inline images
- Fixed type error when creating outlines by utilizing the isString() test

49.21 Version 1.25, 2015-07-07

BUGFIXES:

- Added Python 3 algorithm for ASCII85Decode. Fixes issue when reading reportlab-generated files with Py 3 (jerickbixly)
- Recognize more escape sequence which would otherwise throw an exception (manuelzs, robertsoakes)
- Fixed overflow error in generic.py. Occurred when reading a too-large int in Python 2 (by Raja Jamwal)
- Allow access to files which were encrypted with an empty password. Previously threw a "File has not been decrypted" exception (Elena Williams)
- Do not attempt to decode an empty data stream. Previously would cause an error in decode algorithms (vladir)
- Fixed some type issues specific to Py 2 or Py 3
- Fix issue when stream data begins with whitespace (soloma83)
- Recognize abbreviated filter names (AlmightyOatmeal and Matthew Weiss)

- Copy decryption key from PdfFileReader to PdfFileMerger. Allows usage of PdfFileMerger with encrypted files (twolfson)

- Fixed bug which occurred when a NameObject is present at end of a file stream. Threw a "Stream has ended unexpectedly" exception (speedplane)

FEATURES:

- Initial work on a test suite; to be expanded in future. Tests and Resources directory added, README updated (robertsoakes)

- Added document cloning methods to PdfFileWriter: appendPagesFromReader, cloneReaderDocumentRoot, and cloneDocumentFromReader. See official documentation (robertsoakes)

- Added method for writing to form fields: updatePageFormFieldValues. This will be enhanced in the future. See official documentation (robertsoakes)

- New addAttachment method. See documentation. Support for adding and extracting embedded files to be enhanced in the future (moshekaplan)

- Added methods to get page number of given PageObject or Destination: getPageNumber and getDestinationPageNumber. See documentation (mozbugbox)

OTHER ENHANCEMENTS:

- Enhanced type handling (Brent Amrhein)

- Enhanced exception handling in NameObject (sbywater)

- Enhanced extractText method output (peircej)

- Better exception handling

- Enhanced regex usage in NameObject class (speedplane)

49.22 Version 1.24, 2014-12-31

- Bugfixes for reading files in Python 3 (by Anthony Tuininga and pqqp)

- Appropriate errors are now raised instead of infinite loops (by naure and Cyrus Vafadari)

- Bugfix for parsing number tokens with leading spaces (by Maxim Kamenkov)

- Don't crash on bad /Outlines reference (by eshellman)

- Conform tabs/spaces and blank lines to PEP 8 standards

- Utilize the readUntilRegex method when reading Number Objects (by Brendan Jurd)

- More bugfixes for Python 3 and clearer exception handling

- Fixed encoding issue in merger (with eshellman)

- Created separate folder for scripts

49.23 Version 1.23, 2014-08-11

- Documentation now available at pythonhosted.org

- Bugfix in pagerange.py for when `__init__.__doc__` has no value (by Vladir Cruz)

- Fix typos in OutlinesObject().add() (by shilluc)

- Re-added a missing return statement in a utils.py method

- Corrected viewing mode names (by Jason Scheirer)

- New PdfFileWriter method: addJS() (by vfigueiro)

- New bookmark features: color, boldness, italics, and page fit (by Joshua Arnott)

- New PdfFileReader method: getFields(). Used to extract field information from PDFs with interactive forms. See documentation for details

- Converted README file to markdown format (by Stephen Bussard)

- Several improvements to overall performance and efficiency (by mozbugbox)

- Fixed a bug where geospatial information was not scaling along with its page

- Fixed a type issue and a Python 3 issue in the decryption algorithms (with Francisco Vieira and koba-ninkigumi)

- Fixed a bug causing an infinite loop in the ASCII 85 decoding algorithm (by madmaardigan)

- Annotations (links, comment windows, etc.) are now preserved when pages are merged together

- Used the Destination class in addLink() and addBookmark() so that the page fit option could be properly customized

49.24 Version 1.22, 2014-05-29

- Added .DS_Store to .gitignore (for Mac users) (by Steve Witham)

- Removed `__init__()` implementation in NameObject (by Steve Witham)

- Fixed bug (inf. loop) when merging pages in Python 3 (by commx)

- Corrected error when calculating height in scaleTo()

- Removed unnecessary code from DictionaryObject (by Georges Dubus)

- Fixed bug where an exception was thrown upon reading a NULL string (by speedplane)

- Allow string literals (non-unicode strings in Python 2) to be passed to PdfFileReader

- Allow ConvertFunctionsToVirtualList to be indexed with slices and longs (in Python 2) (by Matt Gilson)

- Major improvements and bugfixes to addLink() method (see documentation in source code) (by Henry Keiter)

- General code clean-up and improvements (with Steve Witham and Henry Keiter)

- Fixed bug that caused crash when comments are present at end of dictionary

49.25 Version 1.21, 2014-04-21

- Fix for when /Type isn't present in the Pages dictionary (by Rob1080)

- More tolerance for extra whitespace in Indirect Objects

- Improved Exception handling

- Fixed error in getHeight() method (by Simon Kaempflein)

- implement use of utils.string_type to resolve Py2-3 compatibility issues

- Prevent exception for multiple definitions in a dictionary (with carlosfunk) (only when strict = False)

- Fixed errors when parsing a slice using pdfcat on command line (by Steve Witham)

- Tolerance for EOF markers within 1024 bytes of the actual end of the file (with David Wolever)

- Added overwriteWarnings parameter to PdfFileReader constructor, if False PyPDF2 will NOT overwrite methods from Python's warnings.py module with a custom implementation.

- Fix NumberObject and NameObject constructors for compatibility with PyPy (Rüdiger Jungbeck, Xavier Dupré, shezadkhan137, Steven Witham)

- Utilize utils.Str in pdf.py and pagerange.py to resolve type issues (by egbutter)

- Improvements in implementing StringIO for Python 2 and BytesIO for Python 3 (by Xavier Dupré)

- Added /x00 to Whitespaces, defined utils.WHITESPACES to clarify code (by Maxim Kamenkov)

- Bugfix for merging 3 or more resources with the same name (by lucky-user)

- Improvements to Xref parsing algorithm (by speedplane)

49.26 Version 1.20, 2014-01-27

- Official Python 3+ support (with contributions from TWAC and cgammans) Support for Python versions 2.6 and 2.7 will be maintained

- Command line concatenation (see pdfcat in sample code) (by Steve Witham)

- New FAQ; link included in README

- Allow more (although unnecessary) escape sequences

- Prevent exception when reading a null object in decoding parameters

- Corrected error in reading destination types (added a slash since they are name objects)

- Corrected TypeError in scaleTo() method

- addBookmark() method in PdfFileMerger now returns bookmark (so nested bookmarks can be created)

- Additions to Sample Code and Sample PDFs

- changes to allow 2up script to work (see sample code) (by Dylan McNamee)

- changes to metadata encoding (by Chris Hiestand)

- New methods for links: addLink() (by Enrico Lambertini) and removeLinks()

- Bugfix to handle nested bookmarks correctly (by Jamie Lentin)

- New methods removeImages() and removeText() available for PdfFileWriter (by Tien Haï)

- Exception handling for illegal characters in Name Objects

49.27 Version 1.19, 2013-10-08

BUGFIXES:

- Removed pop in sweepIndirectReferences to prevent infinite loop (provided by ian-su-sirca)
- Fixed bug caused by whitespace when parsing PDFs generated by AutoCad
- Fixed a bug caused by reading a 'null' ASCII value in a dictionary object (primarily in PDFs generated by AutoCad).

FEATURES:

- Added new folders for PyPDF2 sample code and example PDFs; see README for each folder
- Added a method for debugging purposes to show current location while parsing
- Ability to create custom metadata (by jamma313)
- Ability to access and customize document layout and view mode (by Joshua Arnott)

OTHER:

- Added and corrected some documentation
- Added some more warnings and exception messages
- Removed old test/debugging code

UPCOMING:

- More bugfixes (We have received many problematic PDFs via email, we will work with them)
- Documentation - It's time for PyPDF2 to get its own documentation since it has grown much since the original pyPdf
- A FAQ to answer common questions

49.28 Version 1.18, 2013-08-19

- Fixed a bug where older verions of objects were incorrectly added to the cache, resulting in outdated or missing pages, images, and other objects (from speedplane)
- Fixed a bug in parsing the xref table where new xref values were overwritten; also cleaned up code (from speedplane)
- New method mergeRotatedAroundPointPage which merges a page while rotating it around a point (from speedplane)
- Updated Destination syntax to respect PDF 1.6 specifications (from jamma313)
- Prevented infinite loop when a PdfFileReader object was instantiated with an empty file (from Jerome Nexedi)

Other Changes:

- Downloads now available via PyPI
- Installation through pip library is fixed

49.29 Version 1.17, 2013-07-25

- Removed one (from pdf.py) of the two Destination classes. Both classes had the same name, but were slightly different in content, causing some errors. (from Janne Vanhala)

- Corrected and Expanded README file to demonstrate PdfFileMerger

- Added filter for LZW encoded streams (from Michal Horejsek)

- PyPDF2 issue tracker enabled on Github to allow community discussion and collaboration

49.30 Versions -1.16, -2013-06-30

- Note: This ChangeLog has not been kept up-to-date for a while. Hopefully we can keep better track of it from now on. Some of the changes listed here come from previous versions 1.14 and 1.15; they were only vaguely defined. With the new _version.py file we should have more structured and better documented versioning from now on.

- Defined `PyPDF2.__version__`

- Fixed encrypt() method (from Martijn The)

- Improved error handling on PDFs with truncated streams (from cecilkorik)

- Python 3 support (from kushal-kumaran)

- Fixed example code in README (from Jeremy Bethmont)

- Fixed an bug caused by DecimalError Exception (from Adam Morris)

- Many other bug fixes and features by:

  jeansch Anton Vlasenko Joseph Walton Jan Oliver Oelerich Fabian Henze And any others I missed. Thanks for contributing!

49.31 Version 1.13, 2010-12-04

- Fixed a typo in code for reading a "\b" escape character in strings.

- Improved `__repr__` in FloatObject.

- Fixed a bug in reading octal escape sequences in strings.

- Added getWidth and getHeight methods to the RectangleObject class.

- Fixed compatibility warnings with Python 2.4 and 2.5.

- Added addBlankPage and insertBlankPage methods on PdfFileWriter class.

- Fixed a bug with circular references in page's object trees (typically annotations) that prevented correctly writing out a copy of those pages.

- New merge page functions allow application of a transformation matrix.

- To all patch contributors: I did a poor job of keeping this ChangeLog up-to-date for this release, so I am missing attributions here for any changes you submitted. Sorry! I'll do better in the future.

49.32 Version 1.12, 2008-09-02

- Added support for XMP metadata.

- Fix reading files with xref streams with multiple /Index values.

- Fix extracting content streams that use graphics operators longer than 2 characters. Affects merging PDF files.

49.33 Version 1.11, 2008-05-09

- Patch from Hartmut Goebel to permit RectangleObjects to accept NumberObject or FloatObject values.

- PDF compatibility fixes.

- Fix to read object xref stream in correct order.

- Fix for comments inside content streams.

49.34 Version 1.10, 2007-10-04

- Text strings from PDF files are returned as Unicode string objects when pyPdf determines that they can be decoded (as UTF-16 strings, or as PDFDocEncoding strings). Unicode objects are also written out when necessary. This means that string objects in pyPdf can be either generic.ByteStringObject instances, or generic.TextStringObject instances.

- The extractText method now returns a unicode string object.

- All document information properties now return unicode string objects. In the event that a document provides docinfo properties that are not decoded by pyPdf, the raw byte strings can be accessed with an "_raw" property (ie. title_raw rather than title)

- generic.DictionaryObject instances have been enhanced to be easier to use. Values coming out of dictionary objects will automatically be de-referenced (.getObject will be called on them), unless accessed by the new "raw_get" method. DictionaryObjects can now only contain PdfObject instances (as keys and values), making it easier to debug where non-PdfObject values (which cannot be written out) are entering dictionaries.

- Support for reading named destinations and outlines in PDF files. Original patch by Ashish Kulkarni.

- Stream compatibility reading enhancements for malformed PDF files.

- Cross reference table reading enhancements for malformed PDF files.

- Encryption documentation.

- Replace some "assert" statements with error raising.

- Minor optimizations to FlateDecode algorithm increase speed when using PNG predictors.

49.35 Version 1.9, 2006-12-15

- Fix several serious bugs introduced in version 1.8, caused by a failure to run through our PDF test suite before releasing that version.

- Fix bug in NullObject reading and writing.

49.36 Version 1.8, 2006-12-14

- Add support for decryption with the standard PDF security handler. This allows for decrypting PDF files given the proper user or owner password.

- Add support for encryption with the standard PDF security handler.

- Add new pythondoc documentation.

- Fix bug in ASCII85 decode that occurs when whitespace exists inside the two terminating characters of the stream.

49.37 Version 1.7, 2006-12-10

- Fix a bug when using a single page object in two PdfFileWriter objects.

- Adjust PyPDF to be tolerant of whitespace characters that don't belong during a stream object.

- Add documentInfo property to PdfFileReader.

- Add numPages property to PdfFileReader.

- Add pages property to PdfFileReader.

- Add extractText function to PdfFileReader.

49.38 Version 1.6, 2006-06-06

- Add basic support for comments in PDF files. This allows us to read some ReportLab PDFs that could not be read before.

- Add "auto-repair" for finding xref table at slightly bad locations.

- New StreamObject backend, cleaner and more powerful. Allows the use of stream filters more easily, including compressed streams.

- Add a graphics state push/pop around page merges. Improves quality of page merges when one page's content stream leaves the graphics in an abnormal state.

- Add PageObject.compressContentStreams function, which filters all content streams and compresses them. This will reduce the size of PDF pages, especially after they could have been decompressed in a mergePage operation.

- Support inline images in PDF content streams.

- Add support for using .NET framework compression when zlib is not available. This does not make pyPdf compatible with IronPython, but it is a first step.

- Add support for reading the document information dictionary, and extracting title, author, subject, producer and creator tags.

• Add patch to support NullObject and multiple xref streams, from Bradley Lawrence.

49.39 Version 1.5, 2006-01-28

• Fix a bug where merging pages did not work in "no-rename" cases when the second page has an array of content streams.

• Remove some debugging output that should not have been present.

49.40 Version 1.4, 2006-01-27

• Add capability to merge pages from multiple PDF files into a single page using the PageObject.mergePage function. See example code (README or web site) for more information.

• Add ability to modify a page's MediaBox, CropBox, BleedBox, TrimBox, and ArtBox properties through PageObject. See example code (README or web site) for more information.

• Refactor pdf.py into multiple files: generic.py (contains objects like NameObject, DictionaryObject), filters.py (contains filter code), utils.py (various). This does not affect importing PdfFileReader or PdfFileWriter.

• Add new decoding functions for standard PDF filters ASCIIHexDecode and ASCII85Decode.

• Change url and download_url to refer to new pybrary.net web site.

49.41 Version 1.3, 2006-01-23

• Fix new bug introduced in 1.2 where PDF files with \r line endings did not work properly anymore. A new test suite developed with various PDF files should prevent regression bugs from now on.

• Fix a bug where inheriting attributes from page nodes did not work.

49.42 Version 1.2, 2006-01-23

• Improved support for files with CRLF-based line endings, fixing a common reported problem stating "assertion error: assert line == "%%EOF"".

• Software author/maintainer is now officially a proud married person, which is sure to result in better software. . . somehow.

49.43 Version 1.1, 2006-01-18

• Add capability to rotate pages.

• Improved PDF reading support to properly manage inherited attributes from /Type=/Pages nodes. This means that page groups that are rotated or have different media boxes or whatever will now work properly.

• Added PDF 1.5 support. Namely cross-reference streams and object streams. This release can mangle Adobe's PDFReference16.pdf successfully.

49.44 Version 1.0, 2006-01-17

- First distutils-capable true public release. Supports a wide variety of PDF files that I found sitting around on my system.

- Does not support some PDF 1.5 features, such as object streams, cross-reference streams.

PROJECT GOVERNANCE

This document describes how the pypdf project is managed. It describes the different actors, their roles, and the responsibilities they have.

50.1 Terminology

- The **project** is pypdf - a free and open-source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It includes the code, issues, and discussions on GitHub, and the documentation on ReadTheDocs, the package on PyPI, and the website on GitHub.

- A **maintainer** is a person who has technical permissions to change one or more part of the projects. It is a person who is driven to keep the project running and improving.

- A **contributor** is a person who contributes to the project. That could be through writing code - in the best case through forking and creating a pull request, but that is up to the maintainer. Other contributors describe issues, help to ask questions on existing issues to make them easier to answer, participate in discussions, and help to improve the documentation. Contributors are similar to maintainers, but without technial permissions.

- A **user** is a person who imports pypdf into their code. All pypdf users are developers, but not developers who know the internals of pypdf. They only use the public interface of pypdf. They will likely have less knowledge about PDF than contributors.

- The **community** is all of that - the users, the contributors, and the maintainers.

50.2 Governance, Leadership, and Steering pypdf forward

pypdf is a free and open source project with over 100 contributors and likely (way) more than 1000 users.

As pypdf does not have any formal relationship with any company and no funding, all the work done by the community are voluntary contributions. People don't get paid, but choose to spend their free time to create software of which many more are profiting. This has to be honored and respected.

Despite such a big community, the project was dormant from 2016 to 2022. There were still questions asked, issues reported, and pull requests created. But the maintainer didn't have the time to move pypdf forward. During that time, nobody else stepped up to become the new maintainer.

For this reason, pypdf has the **Benevolent Dictator** governance model. The benevolent dictator is a maintainer with all technical permissions - most importantly the permission to push new pypdf versions on PyPI.

Being benevolent, the benevolent dictator listens for decisions to the community and tries their best to make decisions from which the overall community profits - the current one and the potential future one. Being a dictator, the benevolent dictator always has the power and the right to make decisions on their own - also against some members of the community.

As pypdf is free software, parts of the community can split off (fork the code) and create a new community. This should limit the harm a bad benevolent dictator can do.

50.3 Project Language

The project language is (american) English. All documentation and issues must be written in English to ensure that the community can understand it.

We appreciate the fact that large parts of the community don't have English as their mother tongue. We try our best to understand others - automatic translators might help.

50.4 Expectations

The community can expect the following:

- The **benevolent dictator** tries their best to make decisions from which the overall community profits. The benevolent dictator is aware that his/her decisions can shape the overall community. Once the benevolent dictator notices that she/he doesn't have the time to advance pypdf, he/she looks for a new benevolent dictator. As it is expected that the benevolent dictator will step down at some point of their choice (hopefully before their death), it is NOT a benevolent dictator for life (BDFL).

- Every **maintainer** (including the benevolent dictator) is aware of their permissions and the harm they could do. They value security and ensure that the project is not harmed. They give their technical permissions back if they don't need them any longer. Any long-time contributor can become a maintainer. Maintainers can - and should! - step down from their role when they realize that they can no longer commit that time. Their contribution will be honored in the *History of pypdf* .

- Every **contributor** is aware that the time of maintainers and the benevolent dictator is limited. Short pull requests that briefly describe the solved issue and have a unit test have a higher chance to get merged soon - simply because it's easier for maintainers to see that the contribution will not harm the overall project. Their contributions are documented in the git history and in the public issues. Let us know if you would appriciate something else!

- Every **community member** uses a respectful language. We are all human, we get upset about things we care and other things than what's visible on the internet go on in our live. pypdf does not pay its contributors - keep all of that in mind when you interact with others. We are here because we want to help others.

50.4.1 Issues and Discussions

An issue is any technical description that aims at bringing pypdf forward:

- Bugs tickets: Something went wrong because pypdf developers made a mistake.

- Feature requests: pypdf does not support all features of the PDF specifications. There are certainly also convenience methods that would help users a lot.

- Robustness requests: There are many broken PDFs around. In some cases, we can deal with that. It's kind of a mixture between a bug ticket and a feature request.

- Performance tickets: pypdf could be faster - let us know about your specific scenario.

Any comment that is in those technial descriptions which is not helping the discussion can be deleted. This is especially true for "me too" comments on bugs or "bump" comments for desired features. People can express this with 👍 / 👎 reactions.

Discussions are open. No comments will be deleted there - except if they are clearly unrelated spam or only try to insult people (luckily, the community was very respectful so far)

50.4.2 Releases

The maintainers follow semantic versioning. Most importantly, that means that breaking changes will have a major version bump.

Be aware that unintentional breaking changes might still happen. The pypdf maintainers do their best to fix that in a timely manner - please report such issues!

50.5 People

- Martin Thoma is benevolent dictator since April 2022.

- Maintainers:

    - Matthew Stamy (mstamy2) was the benevolent dictator for a long time. He still is around on GitHub once in a while and has permissions on PyPI and GitHub.

    - Matthew Peveler (MasterOdin) is a maintainer on GitHub.

TAKING OWNERSHIP OF PYPDF

pypdf is currently maintained by me, Martin Thoma. I want to avoid that pypdf ever goes unmaintained again. This document serves as a guide to avoid that if I become unavailable, e.g. due to severe health issues.

This currently is just an abstract scenario. I'm fine and I will likely do this for several more years, but I have seen how projects stand still for many years because of the maintainer becomming inactive.

I've also followed the GitHub Deceased User Policy and added a pre-designated successor.

51.1 What belongs to pypdf?

The resources needed for maintaining pypdf are:

- PyPI: pypdf and PyPDF2
- Github: pypdf (the repository, not the organization)
- ReadTheDocs: pypdf and PyPDF2

51.2 When may somebody take ownership?

No activity in 180 days: If I don't answer e-mails (info@martin-thoma.de) and don't make any commits / merges for half a year, you can consider pypdf "not maintained".

51.3 Who may take ownership?

Preferably, one of the owners of the Github `py-pdf` organization takes care of that.

From my current perspective (Martin Thoma, 27th of August 2023), the following people might be candidates:

- Lucas-C: He maintains fpdf2 and is a py-pdf owner
- pubpub-zz: He is the most active contributor to pypdf
- Matthew Peveler: Less active, but he is very careful about breaking changes and an experienced software developer.
- exiledkingcc: He has contributed the core changes related to encryption.

51.4 How to take ownership?

- PyPI: Follow PEP 541 – Package Index Name Retention
- Github: Talk with one of the other py-pdf organization owners
- ReadTheDocs: Follow the Abandoned projects policy

HISTORY OF PYPDF

52.1 The Origins: pyPdf (2005-2010)

In 2005, Mathieu Fenniak launched pyPdf "as a PDF toolkit..." focused on

- document manipulation: by-page splitting, concatenation, and merging;

- document introspection;

- page cropping; and

- document encryption and decryption.

The last release of PyPI was pyPdf 1.13 in 2010.

52.2 PyPDF2 is born (2011-2016)

At the end of 2011, after consultation with Mathieu and others, Phaseit sponsored PyPDF2 as a fork of pyPdf on GitHub. The initial impetus was to handle a wider range of input PDF instances; Phaseit's commercial work often encounters PDF instances "in the wild" that it needs to manage (mostly concatenate and paginate), but that deviate so much from PDF standards that pyPdf can't read them. PyPDF2 reads a considerably wider range of real-world PDF instances.

Neither pyPdf nor PyPDF2 aims to be universal, that is, to provide all possible PDF-related functionality. Note that the similar-appearing pyfpdf of Mariano Reingart is most comparable to ReportLab, in that both ReportLab and pyfpdf emphasize document generation. Interestingly enough, pyfpdf builds in a basic HTML→PDF converter while PyPDF2 has no knowledge of HTML.

So what is PyPDF2 truly about? Think about popular pdftk for a moment. PyPDF2 does what pdftk does, and it does so within your current Python process, and it handles a wider range of variant PDF formats [explain]. PyPDF2 has its own FAQ to answer other questions that have arisen.

The Reddit /r/python crowd chatted obliquely and briefly about PyPDF2 in March 2012.

The core developer / maintainer was Matthew Stamy.

52.3 PyPDF3 and PyPDF4 (2018 - 2022)

Two approaches were made to get PyPDF2 active again: PyPDF3 and PyPDF4.

PyPDF3 had it's first release in 2018 and its last one in February 2022. It never got the user base from PyPDF2.

PyPDF4 only had one release in 2018.

52.4 PyPDF2: Reborn (2022)

Martin Thoma took over maintenance of PyPDF2 in April 2022. It had over 100 open PRs and 321 open issues.

pubpub-zz was extremely active, especially for text extraction.

Matthew Peveler helped a lot with reviews and general project decisions.

exiledkingcc added support for modern encryption schemes.

52.5 pypdf: Back to the Roots (2023-Today)

In order to make things simpler for beginners, PyPDF2 was merged back into pypdf. Now all lowercase, without a number. We hope that the folks who develop PyPDF3 and PyPDF4 also join us.

Compared to PyPDF2 >= 3.0.0, pypdf >= 3.1.0 now offers:

- AES reading and writing support. Not only with PyCryptoDome, but also with cryptography.
- Text extraction improvements, e.g. for math content. pypdf is now comparable with Tika, pypdfium2, and PyMuPDF
- Annotation support
- Performance Improvements and Bugfixes
- Page Label support

CONTRIBUTORS

pypdf had a lot of contributors since it started with pyPdf in 2005. We are a free software project without any company affiliation. We cannot pay contributors, but we do value their contributions. A lot of time, effort, and expertise went into this project. With this list, we recognize those awesome people

The list is definitely not complete. You can find more contributors via the git history and GitHubs 'Contributors' feature.

53.1 Contributors to the pypdf (formerly pyPdf / PyPDF2) project

- abyesilyurt
- ArkieCoder
- Clauss, Christian
- DL6ER
- ediamondscience
- Ermeson, Felipe
- Freitag, François
- Górny, Michał
- Grillo, Miguel
- Gutteridge, David H.
- Hale, Joseph
- harshhes
- JianzhengLuo
- Karvonen, Harry
- King, Hunter
- Kotler, Mitchell
- KourFrost
- Lightup1
- Majumder, Jonah
- Manini, Lorenzo
- maxbeer99

- McNeil, Karen: Arabic Language Support

- Mérino, Antoine

- nalin-udhaar

- Paternault, Louis

- Perrensen, Olsen

- pilotandy

- Pinheiro, Arthur

- Poddar, Arka

- programmarchy

- pubpub-zz: involved in community development

- Ramos, Leodanis Pozo

- RitchieP | LinkedIn | StackOverflow

- robbiebusinessacc

- Roder, Thomas

- Rogmann, Sascha

- Röthenbacher, Thomas

- sietzeberends

- Stober, Marc

- Stüber, Timo

- Thoma, Martin: Maintainer of pypdf since April 2022. I hope to build a great community with many awesome contributors. LinkedIn | StackOverflow | Blog

- Thomas, Reuben

- WevertonGomes

- Wilson, Huon

- ztravis

53.2 Adding a new contributor

Contributors are:

- Anybody who has an commit in main - no matter how big/small or how many. Also if it's via co-authored-by.

- People who opened helpful issues: (1) Bugs: with complete MCVE (2) Well-described feature requests (3) Potentially some more. The maintainers of pypdf have the last call on that one.

- Community work: This is exceptional. If the maintainers of pypdf see people being super helpful in answering issues / discussions or being very active on Stackoverflow, we also consider them being contributors to pypdf.

Contributors can add themselves or ask via an Github Issue to be added.

Please use the following format:

```
* Last name, First name: 140-characters of text; links to linkedin / github / other␣
↪profiles and personal pages are ok

OR

* GitHub Username: 140-characters of text; links to linkedin / github / other profiles␣
↪and personal pages are ok
```

and add the entry in the alphabetical order. People who . The 140 characters are everything visible after the `Name:`.

Please don't use images.

SCOPE OF PYPDF

What features should pypdf have and which features will it never have?

pypdf aims at making interactions with PDF documents simpler. Core tasks that pypdf can perform are:

- Document manipulation: Splitting, merging, cropping, and transforming the pages of PDF files
- Data Extraction: Extract text and metadata from PDF documents
- Security: Decrypt / encrypt PDF documents

Typical indicators that something should be done by pypdf:

- The task needs in-depth knowledge of the PDF format
- It currently requires a lot of code or even is impossible to do with pypdf
- It's neither mentioned in "belongs in user code" nor in "out of scope"
- It already is in the issue list with the is-feature tag.

The moonshot extensions are features we would like to have, but are currently not able to add (PRs are welcome)

54.1 Belongs in user code

Here are a few indicators that a feature belongs into users code (and not into pypdf):

1. The use-case is very specific. Most people will not encounter the same need.
2. It can be done without knowledge of the PDF specification
3. It cannot be done without (non-pdf) domain knowledge. Anything that is specific to your industry.

54.2 Out of scope

While this list is infinitely long, there are a few topics that are asked multiple times.

Those topics are out of scope for pypdf. They will never be part of pypdf:

1. **Optical Character Recognition (OCR)**: OCR is about extracting text from images. That is very different from the kind of text extraction pypdf is doing. Please note that images can be within PDF documents. In the case of scanned documents, the whole page is an image. Some scanners automatically execute OCR and add a text-layer behind the scanned page. That is something pypdf can use, if it's present. As a rule-of-thumb: If you cannot mark/copy the text, it's likely an image. A noteworthy open source OCR project is tesseract.
2. **Format Conversion**: Converting docx / HTML to PDF or PDF to those formats. You might want to have a look at `pdfkit` and similar projects.

Out of scope for the moment, but might be added if there are enough contributors:

- **Digital Signature Support** (reference ticket): Cryptography is complicated. It's important to get it right. pypdf currently doesn't have enough active contributors to properly add digital signautre support. For the moment, pyhanko seems to be the best choice.

- **PDF Generation from Scratch**: pypdf can manipulate existing PDF documents, add annotations, combine / split / crop / transform. It can add blank pages. But if you want to generate invoices, you might want to have a look at `reportlab` / `fpdf2` or document conversion tools like `pdfkit`.

- **Replacing words within a PDF**: Extracting text from PDF is hard. Replacing text in a reliable way is even harder. For example, one word might be split into multiple tokens. Hence it's not a simple "search and replace" in some cases.

- **(Not) Extracting headers/footers/page numbers**: While you can apply heuristics, there is no way to always make it work. PDF documents simply don't contain the information what a header/footer/page number is.

54.2.1 Library vs Application

It's also worth pointing out that `pypdf` is designed to be a library. It is not an application. That has several implications:

- Execution: pypdf cannot be executed directly, but only be called from within a program written by a pypdf user. In contrast, an application is executed by it's own.

- Dependencies: pypdf should have a minimal set of dependencies and only restrict them where it is strictly necessary. In contrast, applications should be installed in environments which are isolated from other applications. They can pin their dependencies.

If you're looking for a way to interact with PDF files via Shell, you should either write a script using pypdf or use `pdfly`.

PYPDF VS X

pypdf is a free and open source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. pypdf can retrieve text and metadata from PDFs as well.

55.1 PyMuPDF and PikePDF

PyMuPDF is a Python binding to MuPDF and PikePDF is the Python binding to QPDF.

While both are excellent libraries for various use-cases, using them is not always possible even when they support the use-case. Both of them are powered by C libraries which makes installation harder and might cause security concerns. For MuPDF you might also need to buy a commercial license.

A core feature of pypdf is that it's pure Python. That means there is no C dependency. It has been used for over 10 years and for this reason a lot of support via StackOverflow and examples on the internet.

55.2 pypdf

PyPDF2 was merged back into `pypdf`. The development continues at `pypdf`.

55.3 PyPDF3 and PyPDF4

Developing and maintaining open source software is extremely time-intensive and in the case of pypdf not paid at all. Having a continuous support is hard.

pypdf was initially released in 2012 on PyPI and received releases until 2016. From 2016 to 2022 there was no update - but people were still using it.

As pypdf is free software, there were attempts to fork it and continue the development. PyPDF3 was first released in 2018 and still receives updates. PyPDF4 has only one release from 2018.

I (Martin Thoma, the current maintainer of pypdf and PyPDF2), hope that we can bring the community back to one path of development. I deprecated PyPDF2 in favor of pypdf already and pypdf has now more features and a cleaner interface than PyPDF2. See *history of pypdf*.

55.4 pdfminer.six and pdfplumber

`pdfminer.six` is capable of extracting the font size / font weight (bold-ness). It has no capabilities for writing PDF files.

`pdfplumber` is a library focused on extracting data from PDF documents. Since `pdfplumber` is built on top of `pdfminer.six`, there are **no capabilities of exporting or modifying a PDF file** (see #440 (discussions)). However, `pdfplumber` is capable of converting a PDF file into an image, draw lines and rectangles on the image, and save it as an image file. Please note that the image conversion is done via ImageMagick (see `pdfplumber`'s documentation).

The `pdfplumber` community is active in answering questions and the library is maintained as of May 2023.

55.5 pdfrw / pdfrw2

I don't have experience with any of those libraries. Please add a comparison if you know pypdf and `pdfrw`!

Please be aware that there is also `pdfminer` which is not maintained. Then there is `pdfrw2` which doesn't have a large community behind it.

55.6 Document Generation

There are (Python) tools to generate PDF documents. pypdf is not one of them.

55.7 CLI applications

pypdf is a pure Python PDF library. If you're looking for an application which you can use from the terminal, give `pdfly` a shot.

FREQUENTLY-ASKED QUESTIONS

56.1 How is pypdf related to PyPDF2?

PyPDF2 was a fork from the original pyPdf. After several years, the fork was merged back into `pypdf` (now all lowercase).

56.2 Which Python versions are supported?

pypdf 3.0+ supports Python 3.6 and later. PyPDF2 2.0+ supports Python 3.6 and later. PyPDF2 1.27.10 supported Python 2.7 to 3.10.

56.3 Who uses pypdf?

pyPdf is vendored into several projects. That means the code of pyPdf was copied into that project.

Projects that depend on pypdf:

- Camelot: A Python library to extract tabular data from PDFs

- edi: Electronic Data Interchange modules

- amazon-textract-textractor: Analyze documents with Amazon Textract and generate output in multiple formats.

- maigret: Collect a dossier on a person by username from thousands of sites

- deda: tracking Dots Extraction, Decoding and Anonymisation toolkit

- opencanary

- Document Conversions

    - rst2pdf

    - xhtml2pdf

    - doc2text

- pdfalyzer: A PDF analysis tool for visualizing the inner tree-like data structure of a PDF in spectacularly large and colorful diagrams as well as scanning the binary streams embedded in the PDF for hidden potentially malicious content.

56.4 How do I cite pypdf?

In BibTeX format:

```
@misc{pypdf,
 title         = {The {pypdf} library},
 author        = {Mathieu Fenniak and
                  Matthew Stamy and
                  pubpub-zz and
                  Martin Thoma and
                  Matthew Peveler and
                  exiledkingcc and {pypdf Contributors}},
 year          = {2022},
 url           = {https://pypi.org/project/pypdf/}
 note          = {See https://pypdf.readthedocs.io/en/latest/meta/CONTRIBUTORS.html for␣
→all contributors}
}
```

56.5 Which License does pypdf use?

pypdf uses the BSD-3-Clause license, see the LICENSE file.

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p
pypdf.annotations,

INDEX

A