
PyPDF2

Mathieu Fenniak

Dec 12, 2022

USER GUIDE

1	Installation	3
2	Migration Guide: 1.x to 2.x	5
3	Imports and Modules	7
4	Naming Adjustments	9
5	Robustness and strict=False	15
6	Exceptions, Warnings, and Log messages	17
7	Metadata	19
8	Extract Text from a PDF	21
9	Extract Images	25
10	Encryption and Decryption of PDFs	27
11	Merging PDF files	29
12	Cropping and Transforming PDFs	31
13	Adding a Stamp/Watermark to a PDF	45
14	Reading PDF Annotations	49
15	Adding PDF Annotations	51
16	Interactions with PDF Forms	57
17	Streaming Data with PyPDF2	59
18	Reduce PDF Size	61
19	PDF Version Support	63
20	The PdfReader Class	65
21	The PdfWriter Class	71
22	The PdfMerger Class	81

23 The PageObject Class	85
24 The Transformation Class	93
25 The DocumentInformation Class	95
26 The XmpInformation Class	97
27 The Destination Class	101
28 The RectangleObject Class	103
29 The Field Class	105
30 The PageRange Class	107
31 The AnnotationBuilder Class	109
32 The PaperSize Class	113
33 Developer Intro	115
34 The PDF Format	117
35 CMaps	121
36 The Deprecation Process	123
37 Testing	125
38 CHANGELOG	127
39 Changelog of PyPDF2 1.X	151
40 Project Governance	175
41 History of PyPDF2	179
42 Contributors	181
43 PyPDF2 vs X	183
44 Frequently-Asked Questions	185
45 Indices and tables	187
Index	189

PyPDF2 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. PyPDF2 can retrieve text and metadata from PDFs as well.

You can contribute to [PyPDF2 on GitHub](#).

INSTALLATION

There are several ways to install PyPDF2. The most common option is to use pip.

1.1 pip

PyPDF2 requires Python 3.6+ to run.

Typically Python comes with pip, a package installer. Using it you can install PyPDF2:

```
pip install PyPDF2
```

If you are not a super-user (a system administrator / root), you can also just install PyPDF2 for your current user:

```
pip install --user PyPDF2
```

1.1.1 Optional dependencies

PyPDF2 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 PyPDF2[full]
```

Alternatively, you can install just some:

If you plan to use PyPDF2 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 PyPDF2[crypto]
```

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

```
pip install PyPDF2[image]
```

1.2 Python Version Support

Python	3.10	3.9	3.8	3.7	3.6	2.7
PyPDF2>=2.0	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 PyPDF2](#) 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/PyPDF2.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 PyPdf) 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`.

IMPORTS AND MODULES

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

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

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:

```
# 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.

PyPDF2 gives you the option to be strict or not.

PyPDF2 has three core objects and all of them have a `strict` parameter:

- *PdfReader*
- *PdfWriter*
- *PdfMerger*

Choosing `strict=True` means that PyPDF2 will raise an exception if a PDF does not follow the specification.

Choosing `strict=False` means that PyPDF2 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

PyPDF2 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 PyPDF2 can deal with.
- **Warnings** are avoidable issues, such as using deprecated classes / functions / parameters. Another example is missing capabilities of PyPDF2. In those cases, PyPDF2 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 PyPDF2 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

Exceptions need to be caught 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 PyPDF2 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:

```
from PyPDF2 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 `PyPDF2.errors.PyPdfError` if you prefer something more specific.

6.2 Warnings

The `warnings` module allows you to ignore warnings:

```
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. PyPDF2 hopefully is having a reasonable level of log messages, but you can reduce which types of messages you want to see:

```
import logging

logger = logging.getLogger("PyPDF2")
logger.setLevel(logging.ERROR)
```

The `logging` module defines six log levels:

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

METADATA

7.1 Reading metadata

```
from PyPDF2 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

```
from PyPDF2 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 the metadata
writer.add_metadata(
    {
        "/Author": "Martin",
        "/Producer": "Libre Writer",
    }
)

# Save the new PDF to a file
```

(continues on next page)

(continued from previous page)

```
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:

```
from PyPDF2 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:

```
# 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: current transformation matrix, text matrix, font-dictionary and font-size. In most cases the x and y coordinates of the current position are in index 4 and 5 of the current transformation 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 might be wrong.

The function provided in argument `visitor_operand_before` has four arguments: operand, 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 ($y > 50$).

```
from PyPDF2 import PdfReader

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

parts = []

def visitor_body(text, cm, tm, fontDict, fontSize):
    y = tm[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.

```
from PyPDF2 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":
        (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) = (tm[4], tm[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

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 for two lowercase letters ‘f’. Should that be parsed as the Unicode symbol ‘’ 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, every 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. PyPDF2 is no OCR software; it will not be able to detect those failures. PyPDF2 will also never be able to extract text from images.

And finally there are issues that PyPDF2 will deal with. If you find such a text extraction bug, please share the PDF with us so we can work on it!

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.

PyPDF2 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 PyPDF2. 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 PyPDF2 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 PyPDF2 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 PyPDF2 has a clear advantage when it comes to characters which are easy to confuse such as o00ö. **PyPDF2 will never confuse characters.** It just reads what is in the file.

PyPDF2 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.

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.

```
from PyPDF2 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
```


ENCRYPTION AND DECRYPTION OF PDFS

Please see the note in the [installation guide](#) for installing the extra dependencies if interacting with PDFs that use AES.

10.1 Encrypt

Add a password to a PDF (encrypt it):

```
from PyPDF2 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")

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

10.2 Decrypt

Remove the password from a PDF (decrypt it):

```
from PyPDF2 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:
```

(continues on next page)

(continued from previous page)

```
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

11.1 Basic Example

```
from PyPDF2 import PdfMerger

merger = PdfMerger()

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.

11.2 Showing more merging options

```
from PyPDF2 import PdfMerger

merger = PdfMerger()

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)
```

(continues on next page)

(continued from previous page)

```
# Close File Descriptors  
merger.close()  
output.close()
```

CROPPING AND TRANSFORMING PDFS

```
from PyPDF2 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 encryption 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("PyPDF2-output.pdf", "wb") as fp:
    writer.write(fp)
```

12.1 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:

```
from PyPDF2 import PdfWriter, PdfReader

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

(continues on next page)

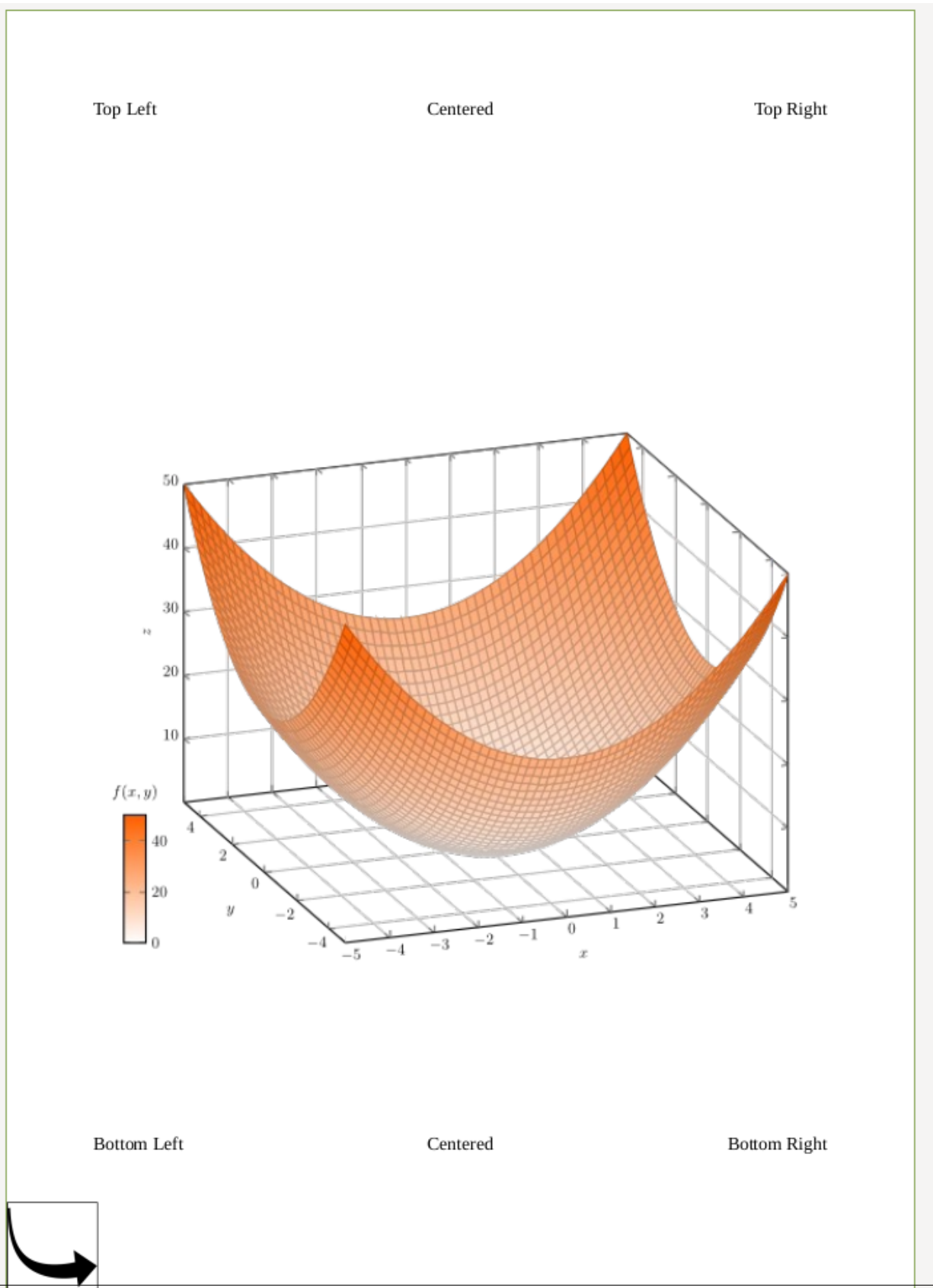
(continued from previous page)

```
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.

12.2 Plain Merge



is the result of

```
from PyPDF2 import PdfReader, PdfWriter, Transformation

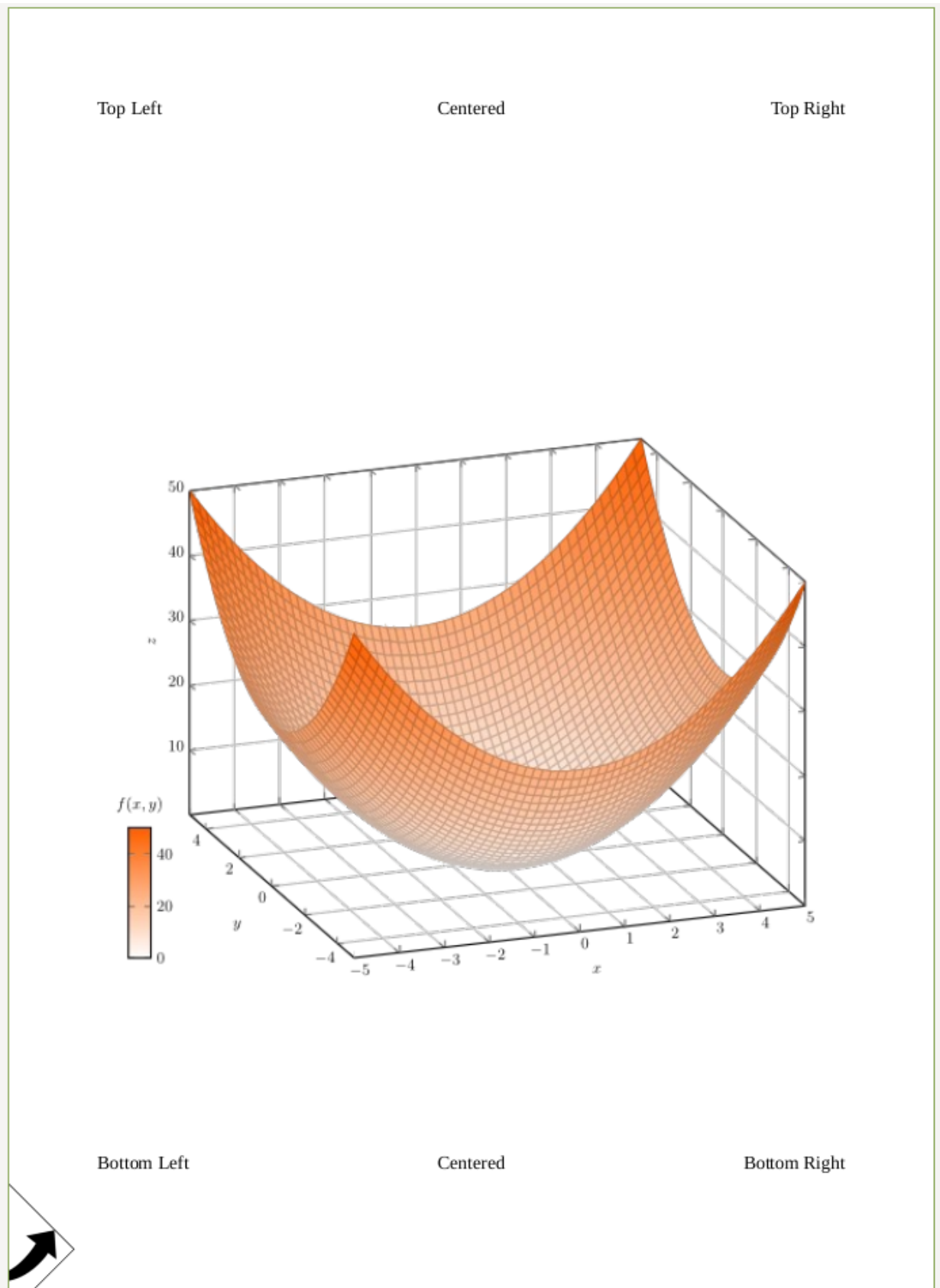
# Get the data
reader_base = PdfReader("labeled-edges-center-image.pdf")
page_base = reader_base.pages[0]

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)
```


12.3 Merge with Rotation



```
from PyPDF2 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]

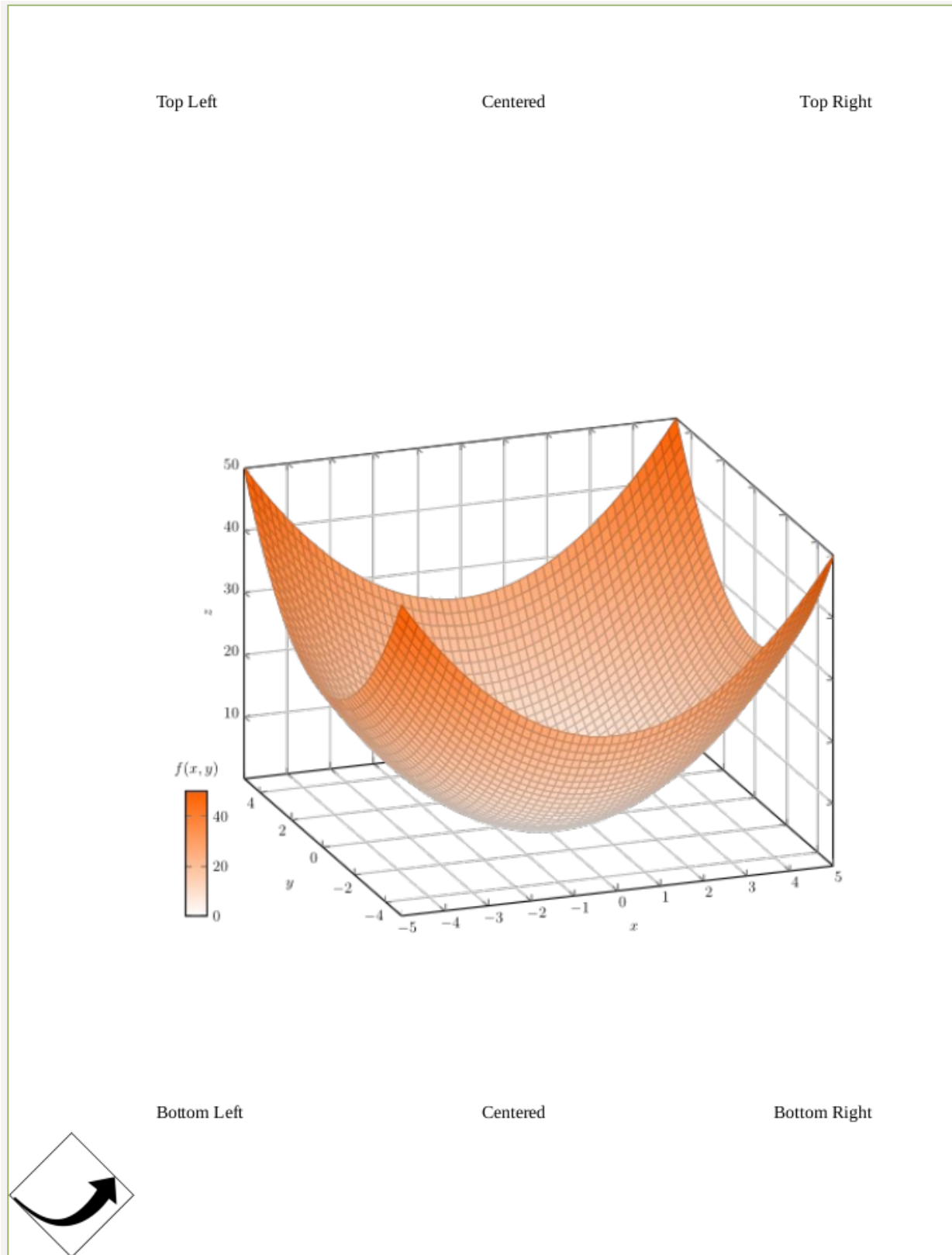
# Apply the transformation
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)

# 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:

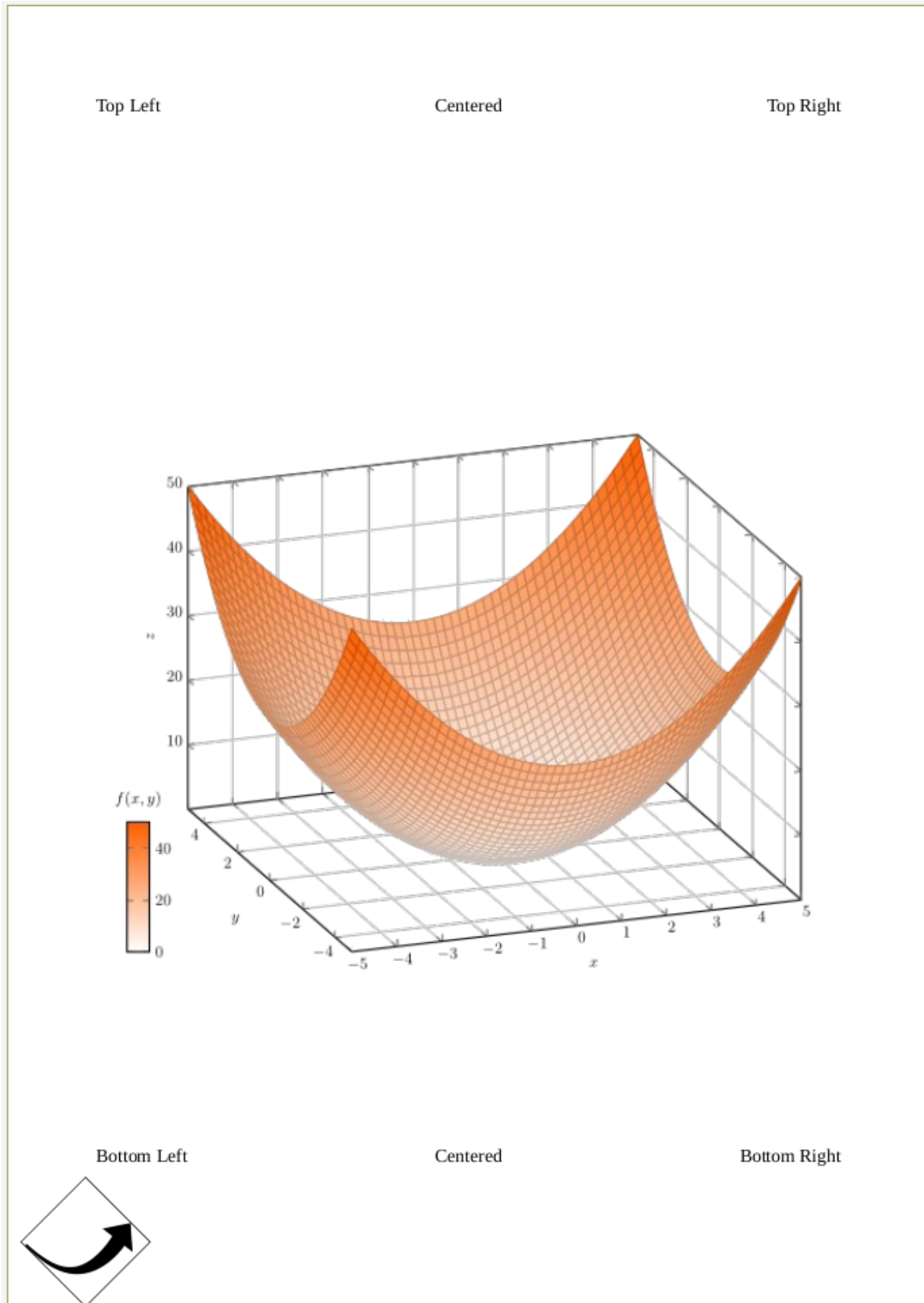
```
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)
```

you get:



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

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



12.4 Scaling

PyPDF2 offers two ways to scale: The page itself and the contents on a page. Typically, you want to combine both.



12.4.1 Scaling a Page (the Canvas)

```
from PyPDF2 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:

```
from PyPDF2.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))
```

12.4.2 Scaling the content

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

```
from PyPDF2 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")
```


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.

In both cases you might want to ensure that the mediabox/cropbox of the original content stays the same.

13.1 Stamp (Overlay)

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

from PyPDF2 import PdfWriter, PdfReader

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

    writer = PdfWriter()

    reader = PdfReader(content_pdf)
    if page_indices == "ALL":
        page_indices = list(range(0, len(reader.pages)))
    for index in page_indices:
        content_page = reader.pages[index]
        mediabox = content_page.mediabox
        content_page.merge_page(image_page)
        content_page.mediabox = mediabox
        writer.add_page(content_page)

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

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 see so
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 must make tools for these kinds of people.

Whats some say them as the crazy ones, were genius. Because the
people who are crazy enough to think they can change the world,
are the ones who do.

13.2 Watermark (Underlay)

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

from PyPDF2 import PdfWriter, PdfReader

def watermark(
    content_pdf: Path,
    stamp_pdf: Path,
    pdf_result: Path,
    page_indices: Union[Literal["ALL"], List[int]] = "ALL",
):
    reader = PdfReader(content_pdf)
```

(continues on next page)

(continued from previous page)

```
if page_indices == "ALL":
    page_indices = list(range(0, len(reader.pages)))

writer = PdfWriter()
for index in page_indices:
    content_page = reader.pages[index]
    mediabox = content_page.mediabox

    # You need to load it again, as the last time it was overwritten
    reader_stamp = PdfReader(stamp_pdf)
    image_page = reader_stamp.pages[0]

    image_page.merge_page(content_page)
    image_page.mediabox = mediabox
    writer.add_page(image_page)

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

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.

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:

```
from PyPDF2 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.

14.1 Text

```
from PyPDF2 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"])
```

14.2 Highlights

```
from PyPDF2 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
```

14.3 Attachments

```
from PyPDF2 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

15.1 Attachments

```
from PyPDF2 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)
```

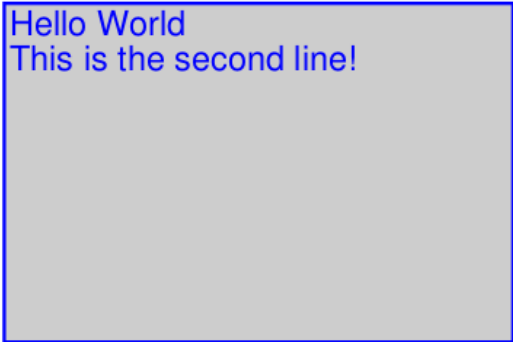
15.2 Free Text

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 *AnnotationBuilder*:

```
from PyPDF2 import PdfReader, PdfWriter
from PyPDF2.generic import AnnotationBuilder

# 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 = AnnotationBuilder.free_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)


(continued from previous 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)
```

15.3 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

15.4 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 the [AnnotationBuilder](#):

```
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 = AnnotationBuilder.line(
    text="Hello World\nLine2",
    rect=(50, 550, 200, 650),
    p1=(50, 550),
    p2=(200, 650),
)
writer.add_annotation(page_number=0, annotation=annotation)

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

(continues on next page)

(continued from previous page)

```
writer.write(fp)
```

15.5 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 the [AnnotationBuilder](#):

```
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 = AnnotationBuilder.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.

15.6 Link

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

```
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 = AnnotationBuilder.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:

```
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 = AnnotationBuilder.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)
```

INTERACTIONS WITH PDF FORMS

16.1 Reading form fields

```
from PyPDF2 import PdfReader

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

16.2 Filling out forms

```
from PyPDF2 import PdfReader, PdfWriter

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

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

writer.add_page(page)

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

# write "output" to PyPDF2-output.pdf
with open("filled-out.pdf", "wb") as output_stream:
    writer.write(output_stream)
```


STREAMING DATA WITH PYPDF2

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.

PyPDF2 supports streaming data to a file-like object and here is how.

```
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)
```

17.1 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`:

```
from io import BytesIO

import boto3
from PyPDF2 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)

(continued from previous page)

```
# 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
    )
```


REDUCE PDF 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.

18.1 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:

```
from PyPDF2 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.

18.2 Remove images

```
from PyPDF2 import PdfReader, PdfWriter

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

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

writer.remove_images()
```

(continues on next page)

(continued from previous page)

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

18.3 Lossless Compression

PyPDF2 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`:

```
from PyPDF2 import PdfReader, PdfWriter

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

for page in reader.pages:
    page.compress_content_streams() # This is CPU intensive!
    writer.add_page(page)

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

Using this method, we have seen a reduction by 70% (from 11.8 MB to 3.5 MB) with a real 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 PyPDF2 can do the operations you want on PDF 2.0 files without fully supporting all features of PDF 2.0.

19.1 PDF Feature Support by PyPDF2

Feature	PDF-Version	PyPDF2 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 PyPDF2, but other libraries can be used for them:

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

THE PDFREADER CLASS

```
class PyPDF2.PdfReader(stream: Union[str, BytesIO, BufferedReader, BufferedWriter, FileIO, 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** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to `False`.
- **password** (*None/str/bytes*) – Decrypt PDF file at initialization. If the password is `None`, the file will not be decrypted. Defaults to `None`

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

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

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

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

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

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

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

```
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 (*str*) – The password to match.

Returns

PasswordType.

property documentInfo: `Optional[DocumentInformation]`

Deprecated since version 1.28.0.

Use the attribute `metadata` instead.

getDestinationPageNumber(*destination: Destination*) → int

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

getDocumentInfo() → `Optional[DocumentInformation]`

Deprecated since version 1.28.0: Use the attribute `metadata` instead.

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

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

getFormTextFields() → `Dict[str, Any]`

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

getIsEncrypted() → bool

Deprecated since version 1.28.0: Use `is_encrypted` instead.

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

Deprecated since version 1.28.0: Use `named_destinations` instead.

getNumPages() → int

Deprecated since version 1.28.0: Use `len(reader.pages)` instead.

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

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

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

Deprecated since version 1.28.0: Use `outline` instead.

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

Deprecated since version 1.28.0: Use `reader.pages[page_number]` instead.

getPageLayout() → `Optional[str]`

Deprecated since version 1.28.0: Use `page_layout` instead.

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

Deprecated since version 1.28.0: Use `page_mode` instead.

getPageNumber(*page: PageObject*) → int

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

getXmpMetadata() → `Optional[XmpInformation]`

Deprecated since version 1.28.0: Use the attribute `xmp_metadata` instead.

get_destination_page_number(*destination: Destination*) → int

Retrieve page number of a given Destination object.

Parameters

destination (`Destination`) – The destination to get page number.

Returns

the page number or -1 if page not found

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

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.

get_form_text_fields() → Dict[str, Any]

Retrieve form fields from the document with textual data.

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*, ...

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

get_page_number(*page: PageObject*) → int

Retrieve page number of a given PageObject

Parameters

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

Returns

the page number or -1 if page not found

property isEncrypted: bool

Deprecated since version 1.28.0.

Use *is_encrypted* instead.

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.

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.

Returns

the document information of this PDF file

property namedDestinations: Dict[str, Any]

Deprecated since version 1.28.0.

Use *named_destinations* instead.

property named_destinations: Dict[str, Any]

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

property numPages: int

Deprecated since version 1.28.0.

Use *len(reader.pages)* instead.

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

Read-only property for the outline (i.e., a collection of ‘outline items’ which are also known as ‘bookmarks’) present in the document.

Returns

a nested list of *Destinations*.

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

Deprecated since version 2.9.0.

Use *outline* instead.

property pageLayout: `Optional[str]`

Deprecated since version 1.28.0.

Use *page_layout* instead.

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

Deprecated since version 1.28.0.

Use *page_mode* instead.

property page_layout: `Optional[str]`

Get the page layout.

Returns

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

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

Get the page mode.

Returns

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

property pages: `List[PageObject]`

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

property pdf_header: `str`

read(*stream*: `Union[BytesIO, BufferedReader, BufferedWriter, FileIO]`) \rightarrow `None`

readNextEndLine(*stream*: `Union[BytesIO, BufferedReader, BufferedWriter, FileIO]`, *limit_offset*: `int = 0`) \rightarrow `bytes`

Deprecated since version 1.28.0.

readObjectHeader(*stream*: `Union[BytesIO, BufferedReader, BufferedWriter, FileIO]`) \rightarrow `Tuple[int, int]`

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

read_next_end_line(*stream*: `Union[BytesIO, BufferedReader, BufferedWriter, FileIO]`, *limit_offset*: `int = 0`) \rightarrow `bytes`

Deprecated since version 2.1.0.

read_object_header(*stream*: `Union[BytesIO, BufferedReader, BufferedWriter, FileIO]`) \rightarrow `Tuple[int, int]`

property threads: `Optional[ArrayObject]`

Read-only property for the list of threads see §8.3.2 from PDF 1.7 spec

Returns

an Array of Dictionnaires with “/F” and “/I” properties or None if no articles.

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

property xmpMetadata: `Optional[XmpInformation]`

Deprecated since version 1.28.0.

Use the attribute `xmp_metadata` instead.

property xmp_metadata: `Optional[XmpInformation]`

XMP (Extensible Metadata Platform) data

Returns

a `XmpInformation` instance that can be used to access XMP metadata from the document.
or None if no metadata was found on the document root.

THE PDFWRITER CLASS

```
class PyPDF2.PdfWriter(fileobj: Union[str, BytesIO, BufferedReader, BufferedWriter, FileIO] = "")
```

Bases: object

This class supports writing PDF files out, given pages produced by another class (typically *PdfReader*).

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

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

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

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

```
addBookmark(title: str, pagenum: int, parent: ~typing.Union[None, ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] = None, color: ~typing.Optional[~typing.Tuple[float, float, float]] = None, bold: bool = False, italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float]) → IndirectObject
```

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

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

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

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

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

```
addJS(javascript: str) → None
```

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

```
addLink(pagenum: int, pagedest: int, rect: ~PyPDF2.generic._rectangle.RectangleObject, border: ~typing.Optional[~PyPDF2.generic._data_structures.ArrayObject] = None, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float]) → None
```

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

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

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

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

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

addNamedDestinationObject(*dest: PdfObject*) → IndirectObject

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

addPage(*page: PageObject*) → None

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

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

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

add_annotation(*page_number: int, annotation: Dict[str, Any]*) → None

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

Embed a file inside the PDF.

Parameters

- **filename** (*str*) – The filename to display.
- **data** (*str*) – The data in the file.

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

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** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – 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.

add_bookmark(*title: str, pagenum: int, parent: ~typing.Union[None, ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] = None, color: ~typing.Optional[~typing.Tuple[float, float, float]] = None, bold: bool = False, italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float])* → IndirectObject

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

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

Deprecated since version 2.9.0: Use `add_outline_item_destination()` instead.

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

Deprecated since version 2.9.0: Use `add_outline_item_dict()` instead.

add_js(*javascript: str*) → None

Add Javascript which will launch upon opening this PDF.

Parameters

javascript (*str*) – 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.
```

add_link(*pagenum*: int, *pagedest*: int, *rect*: ~PyPDF2.generic._rectangle.RectangleObject, *border*: ~typing.Optional[~PyPDF2.generic._data_structures.ArrayObject] = None, *fit*: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = 'Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float]) → None

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

Add custom metadata to the output.

Parameters

infos (dict) – a Python dictionary where each key is a field and each value is your new metadata.

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

add_named_destination_object(*dest*: PdfObject) → IndirectObject

add_outline() → None

add_outline_item(*title*: str, *page_number*: ~typing.Optional[int] = None, *parent*: ~typing.Union[None, ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] = None, *color*: ~typing.Optional[~typing.Union[~typing.Tuple[float, float, float], str]] = None, *bold*: bool = False, *italic*: bool = False, *fit*: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = 'Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float], *pagenum*: ~typing.Optional[int] = None) → IndirectObject

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

Parameters

- **title** (str) – Title to use for this outline item.
- **page_number** (int) – Page number this outline item will point to.
- **parent** – A reference to a parent outline item to create nested outline items.
- **color** (tuple) – 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** (bool) – Outline item font is bold
- **italic** (bool) – Outline item font is italic
- **fit** (str) – The fit of the destination page. See [add_link\(\)](#) for details.

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

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

add_page(page: [PageObject](#)) → None

Add a page to this PDF file.

The page is usually acquired from a [PdfReader](#) instance.

Parameters

page ([PageObject](#)) – The page to add to the document. Should be an instance of [PageObject](#)

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** (int) – index of the page on which to place the URI action.
- **uri** (str) – URI of resource to link to.
- **rect** (Tuple[int, int, int, int]) – [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** (ArrayObject) – if provided, an array describing border-drawing properties. See the PDF spec for details. No border will be drawn if this argument is omitted.

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

Deprecated since version 1.28.0: Use [append_pages_from_reader\(\)](#) instead.

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.

Parameters

- **reader** ([PdfReader](#)) – a PdfReader object from which to copy page annotations to this writer object. The writer's annots will then be updated
- **after_page_append** (Callable[[[PageObject](#)], None]) – 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

Deprecated since version 1.28.0: Use [clone_document_from_reader\(\)](#) instead.

cloneReaderDocumentRoot(reader: [PdfReader](#)) → None

Deprecated since version 1.28.0: Use [clone_reader_document_root\(\)](#) instead.

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

Parameters

- **reader** – PDF file reader instance from which the clone should be created.

- **after_page_append** (*Callable*[[*PageObject*], *None*]) – 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.

clone_reader_document_root(*reader*: *PdfReader*) → *None*

Copy the reader document root to the writer.

Parameters

reader – PdfReader from the document root should be copied.

encrypt(*user_pwd*: *str*, *owner_pwd*: *typing.Optional[str]* = *None*, *use_128bit*: *bool* = *True*, *permissions_flag*: *~PyPDF2.constants.UserAccessPermissions* = *UserAccessPermissions.None*) → *None*

Encrypt this PDF file with the PDF Standard encryption handler.

Parameters

- **user_pwd** (*str*) – The “user password”, which allows for opening and reading the PDF file with the restrictions provided.
- **owner_pwd** (*str*) – The “owner 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** (*bool*) – flag as to whether to use 128bit encryption. When false, 40bit encryption will be used. By default, this flag is on.
- **permissions_flag** (*unsigned int*) – permissions as described in TABLE 3.20 of the PDF 1.7 specification. A bit value of 1 means the permission is granted. 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.

getNamedDestRoot() → *ArrayObject*

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

getNumPages() → *int*

Deprecated since version 1.28.0: Use `len(writer.pages)` instead.

getObject(*ido*: *IndirectObject*) → *PdfObject*

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

getOutlineRoot() → *TreeObject*

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

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

Deprecated since version 1.28.0: Use `writer.pages[page_number]` instead.

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

Deprecated since version 1.28.0: Use `page_layout` instead.

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

Deprecated since version 1.28.0: Use `page_mode` instead.

getReference(*obj*: *PdfObject*) → *IndirectObject*

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

get_named_dest_root() → *ArrayObject*

get_object(*ido: IndirectObject*) → PdfObject

get_outline_root() → TreeObject

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

Retrieve a page by number from this PDF file.

Parameters

page_number (*int*) – The page number to retrieve (pages begin at zero)

Returns

the page at the index given by *page_number*

get_reference(*obj: PdfObject*) → IndirectObject

get_threads_root() → ArrayObject

the list of threads see §8.3.2 from PDF 1.7 spec

return

an Array (possibly empty) of Dictionaries with “/F” and “/T” properties

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

Deprecated since version 1.28.0: Use [insertBlankPage\(\)](#) instead.

insertPage(*page: PageObject, index: int = 0*) → None

Deprecated since version 1.28.0: Use [insert_page\(\)](#) instead.

insert_blank_page(*width: Optional[Decimal] = None, height: Optional[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** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – The height of the new page expressed in default user space units.
- **index** (*int*) – 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.

insert_page(*page: PageObject, index: int = 0*) → None

Insert a page in this PDF file. The page is usually acquired from a [PdfReader](#) instance.

Parameters

- **page** ([PageObject](#)) – The page to add to the document.
- **index** (*int*) – Position at which the page will be inserted.

property open_destination: Union[None, [Destination](#), TextStringObject, 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.

:param destination: the property can be set to a Destination, a Page or an string(NamedDest) or

None (to remove “/OpenAction”)

(value stored in “/OpenAction” entry in the Pdf Catalog)

property `pageLayout`: `Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]`

Deprecated since version 1.28.0.

Use `page_layout` instead.

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

Deprecated since version 1.28.0.

Use `page_mode` instead.

property `page_layout`: `Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]`

Page layout property.

Table 1: Valid layout values

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

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

Page mode property.

Table 2: Valid mode values

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

property `pages`: `List[PageObject]`

Property that emulates a list of `PageObject`.

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.

`removeImages(ignoreByteStringObject: bool = False)` \rightarrow None

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

removeLinks() → None

Deprecated since version 1.28.0: Use [remove_links\(\)](#) instead.

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

Deprecated since version 1.28.0: Use [remove_text\(\)](#) instead.

remove_images(*ignore_byte_string_object: bool = False*) → None

Remove images from this output.

Parameters

ignore_byte_string_object (*bool*) – optional parameter to ignore ByteString Objects.

remove_links() → None

Remove links and annotations from this output.

remove_text(*ignore_byte_string_object: bool = False*) → None

Remove text from this output.

Parameters

ignore_byte_string_object (*bool*) – optional parameter to ignore ByteString Objects.

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

Deprecated since version 1.28.0: Use [page_layout](#) instead.

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

Deprecated since version 1.28.0: Use [page_mode](#) instead.

set_need_appearances_writer() → None

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

Deprecated since version 1.28.0: Use [page_mode](#) instead.

property threads: ArrayObject

Read-only property for 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

updatePageFormFieldValues(*page: ~PyPDF2._page.PageObject, fields: ~typing.Dict[str, ~typing.Any], flags: ~PyPDF2.constants.FieldFlag = FieldFlag.None*) → None

Deprecated since version 1.28.0: Use [update_page_form_field_values\(\)](#) instead.

update_page_form_field_values(*page: ~PyPDF2._page.PageObject, fields: ~typing.Dict[str, ~typing.Any], flags: ~PyPDF2.constants.FieldFlag = FieldFlag.None*) → 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** ([PageObject](#)) – Page reference from PDF writer where the annotations and field data will be updated.
- **fields** (*dict*) – a Python dictionary of field names (/T) and text values (/V)

- **flags** (*int*) – 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.

write(*stream: Union[Path, str, BytesIO, BufferedReader, BufferedWriter, FileIO]*) → Tuple[bool, Union[FileIO, BytesIO, BufferedReader, BufferedWriter]]

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.

write_stream(*stream: Union[BytesIO, BufferedReader, BufferedWriter, FileIO]*) → None

THE PDFMERGER CLASS

```
class PyPDF2.PdfMerger(strict: bool = False, fileobj: Union[Path, str, BytesIO, BufferedReader, BufferedWriter,
    FileIO] = "")
```

Bases: object

Initialize a PdfMerger object.

PdfMerger merges multiple PDFs into a single PDF. It can concatenate, slice, insert, or any combination of the above.

See the functions [merge\(\)](#) (or [append\(\)](#)) and [write\(\)](#) for usage information.

Parameters

- **strict** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to False.
- **fileobj** – Output file. Can be a filename or any kind of file-like object.

```
addBookmark(title: str, pagenum: int, parent: ~typing.Union[None,
    ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] =
    None, color: ~typing.Optional[~typing.Tuple[float, float, float]] = None, bold: bool = False,
    italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH,
    /FitBV] = '/Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject,
    ~PyPDF2.generic._base.NullObject, float]) → IndirectObject
```

Deprecated since version 1.28.0: Use [add_outline_item\(\)](#) instead.

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

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

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

Deprecated since version 1.28.0: Use [add_named_destination\(\)](#) instead.

```
add_bookmark(title: str, pagenum: int, parent: ~typing.Union[None,
    ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] =
    None, color: ~typing.Optional[~typing.Tuple[float, float, float]] = None, bold: bool = False,
    italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH,
    /FitBV] = '/Fit', *args: ~typing.Union[~PyPDF2.generic._base.NumberObject,
    ~PyPDF2.generic._base.NullObject, float]) → IndirectObject
```

Deprecated since version 2.9.0: Use [add_outline_item\(\)](#) instead.

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

Add custom metadata to the output.

Parameters

infos (*dict*) – a Python dictionary where each key is a field and each value is your new metadata. Example: `{u'/Title': u'My title'}`

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

Add a destination to the output.

Parameters

- **title** (str) – Title to use
- **page_number** (int) – Page number this destination points at.

add_outline_item(title: str, page_number: ~typing.Optional[int] = None, parent: ~typing.Union[None, ~PyPDF2.generic._data_structures.TreeObject, ~PyPDF2.generic._base.IndirectObject] = None, color: ~typing.Optional[~typing.Tuple[float, float, float]] = None, bold: bool = False, italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = /Fit, *args: ~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float], pagenum: ~typing.Optional[int] = None) → IndirectObject

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

Parameters

- **title** (str) – Title to use for this outline item.
- **page_number** (int) – Page number this outline item will point to.
- **parent** – A reference to a parent outline item to create nested outline items.
- **color** (tuple) – Color of the outline item’s font as a red, green, blue tuple from 0.0 to 1.0
- **bold** (bool) – Outline item font is bold
- **italic** (bool) – Outline item font is italic
- **fit** (str) – The fit of the destination page. See `add_link()` for details.

append(fileobj: Union[str, BytesIO, BufferedReader, BufferedWriter, FileIO, 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** (str) – 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** (bool) – 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.

close() → None

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

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.

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

merge(*position*: int, *fileobj*: Union[Path, str, BytesIO, BufferedReader, BufferedWriter, FileIO, PdfReader], *outline_item*: Optional[str] = None, *pages*: Optional[Union[str, PageRange, Tuple[int, int], Tuple[int, int, int], List[int]]] = None, *import_outline*: bool = True) → None

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

Parameters

- **position** (int) – 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** (str) – 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** (bool) – 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.

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

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

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

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

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

Set the page layout.

Parameters

layout (str) – 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

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

Set the page mode.

Parameters

mode (*str*) – The page mode to use.

Table 2: Valid **mode** arguments

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

write(*fileobj*: *Union*[*Path*, *str*, *BytesIO*, *BufferedReader*, *BufferedWriter*, *FileIO*]) → *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.

THE PAGEOBJECT CLASS

```
class PyPDF2._page.PageObject(pdf: Optional[Any] = 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_ref** – Stores the original indirect reference to this object in its source PDF

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

Deprecated since version 1.28.0: Use *add_transformation()* 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 (*tuple*) – A 6-element tuple containing the operands of the transformation matrix. Alternatively, a *Transformation* object can be passed.

See *Cropping and Transforming PDFs*.

```
property annotations: Optional[ArrayObject]
```

```
property artBox: RectangleObject
```

Deprecated since version 1.28.0.

Use *artbox* instead.

```
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 bleedBox: RectangleObject
```

Deprecated since version 1.28.0.

Use *bleedbox* instead.

```
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.

compressContentStreams() → None

Deprecated since version 1.28.0: Use [compress_content_streams\(\)](#) instead.

compress_content_streams() → 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”.

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

Deprecated since version 1.28.0: Use [create_blank_page\(\)](#) instead.

static create_blank_page(pdf: Optional[Any] = 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** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – 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

property cropBox: [RectangleObject](#)

Deprecated since version 1.28.0.

Use [cropbox](#) instead.

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](#).

extractText(Tj_sep: str = ", TJ_sep: str = ") → str

Deprecated since version 1.28.0: Use [extract_text\(\)](#) instead.

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.

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 PyPDF2==4.0.0
- **TJ_sep** – Deprecated. Kept for compatibility until PyPDF2==4.0.0
- **orientations** – (list of) orientations (of the characters) (default: (0,90,270,360)) single int is equivalent to a singleton (0 == (0,)) note: currently only 0(Up),90(turned Left), 180(upside Down),270 (turned Right)
- **space_width** (*float*) – force default space width (if not extracted from font (default: 200))
- **visitor_operand_before** (*Optional[Function]*) – function to be called before processing an operand. It has four arguments: operator, operand-arguments, current transformation matrix and text matrix.
- **visitor_operand_after** (*Optional[Function]*) – function to be called after processing an operand. It has four arguments: operand, operand-arguments, current transformation matrix and text matrix.
- **visitor_text** (*Optional[Function]*) – function to be called when extracting some text at some position. It has three arguments: text, current transformation matrix and text matrix.

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

space_width (*float*) – force default space width (if not extracted from font (default 200))

Returns

The extracted text

getContents() → Optional[ContentStream]

Deprecated since version 1.28.0: Use [get_contents\(\)](#) instead.

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

hash_value_data() → bytes

property images: List[File]

Get a list of all images of the page.

This requires pillow. You can install it via 'pip install PyPDF2[image]'.

For the moment, this does NOT include inline images. They will be added in future.

property mediaBox: [*RectangleObject*](#)

Deprecated since version 1.28.0.

Use [*mediabox*](#) instead.

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.

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

Deprecated since version 1.28.0: Use [*merge_page\(\)*](#) instead.

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

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

Parameters

- **page2** ([*PageObject*](#)) – the page to be merged into this one. Should be an instance of [*PageObject*](#).
- **rotation** (*float*) – The angle of the rotation, in degrees
- **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.

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

mergeRotatedScaledPage is similar to *merge_page*, but the stream to be merged is 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*](#).
- **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.

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.

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

`mergeRotatedTranslatedPage` is similar to `merge_page`, but the stream to be merged is rotated and translated 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
- **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.

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

`mergeScaledPage` is similar to `merge_page`, but the stream to be merged is 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
- **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.

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.

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

mergeTransformedPage is similar to merge_page, but a transformation matrix is applied to the merged stream.

Parameters

- **page2** ([PageObject](#)) – The page to be merged into this one. Should be an instance of [PageObject](#).
- **ctm** (*tuple*) – a 6-element tuple containing the operands of the transformation matrix
- **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.

mergeTranslatedPage(page2: [PageObject](#), tx: float, ty: float, 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** ([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
- **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.

merge_page(page2: [PageObject](#), expand: bool = False) → 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** ([PageObject](#)) – The page to be merged into this one. Should be an instance of [PageObject](#).
- **expand** (*bool*) – If true, the current page dimensions will be expanded to accommodate the dimensions of the page to be merged.

rotate(angle: int) → [PageObject](#)

Rotate a page clockwise by increments of 90 degrees.

Parameters

angle (*int*) – Angle to rotate the page. Must be an increment of 90 deg.

rotateClockwise(angle: int) → [PageObject](#)

Deprecated since version 1.28.0: Use [rotate_clockwise\(\)](#) instead.

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

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

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

property rotation: int

The VISUAL rotation of the page.

This number has to be a multiple of 90 degrees: 0,90,180,270 This property does not affect “/Contents”

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** (*float*) – The scaling factor on horizontal axis.
- **sy** (*float*) – The scaling factor on vertical axis.

scaleBy(*factor: float*) → None

Deprecated since version 1.28.0: Use [scale_by\(\)](#) instead.

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

Deprecated since version 1.28.0: Use [scale_to\(\)](#) instead.

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

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** (*float*) – The new width.
- **height** (*float*) – The new height.

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.

property trimBox: [RectangleObject](#)

Deprecated since version 1.28.0.

Use [trimbox](#) instead.

property trimbox

A [RectangleObject](#), expressed in default user space units, defining the intended dimensions of the finished page after trimming.

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.

THE TRANSFORMATION CLASS

```
class PyPDF2.Transformation(ctm: Tuple[float, float, float, float, float, float] = (1, 0, 0, 1, 0, 0))
```

Bases: object

Specify a 2D transformation.

The transformation between two coordinate systems is represented by a 3-by-3 transformation matrix written as follows:

```
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:

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

24.1 Usage

```
>>> from PyPDF2 import Transformation
>>> op = Transformation().scale(sx=2, sy=3).translate(tx=10, ty=20)
>>> page.add_transformation(op)
```

```
apply_on(pt: Union[Tuple[Decimal, Decimal], Tuple[float, float], List[float]]) → Union[Tuple[float, float],
List[float]]
```

```
static compress(matrix: Tuple[Tuple[float, float, float], Tuple[float, float, float], Tuple[float, float, float]])
→ Tuple[float, float, float, float, float, float]
```

```
property matrix: Tuple[Tuple[float, float, float], Tuple[float, float, float],
Tuple[float, float, float]]
```

```
rotate(rotation: float) → Transformation
```

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.

translate(*tx: float = 0, ty: float = 0*) → *Transformation*

THE DOCUMENTINFORMATION CLASS

class PyPDF2.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 PyPDF2 was unable to decode the string's text encoding; this requires additional safety in the caller and therefore is not as commonly accessed.

property `author`: `Optional[str]`

Read-only property accessing the document's **author**.

Returns a unicode string (`TextStringObject`) or `None` if the author is not specified.

property `author_raw`: `Optional[str]`

The "raw" version of author; 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[+-]hh'mm` where the suffix is the offset from UTC.

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 unicode string (`TextStringObject`) or `None` if the creator is not specified.

property `creator_raw`: `Optional[str]`

The "raw" version of creator; can return a `ByteStringObject`.

getText (*key*: *str*) → `Optional[str]`

The text value of the specified key or `None`.

Deprecated since version 1.28.0: Use the attributes (e.g. [*title*](#) / [*author*](#)).

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[+-]hh'mm` where the suffix is the offset from UTC.

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 unicode string (`TextStringObject`) or `None` if the producer is not specified.

property producer_raw: Optional[str]

The “raw” version of producer; can return a `ByteStringObject`.

property subject: Optional[str]

Read-only property accessing the document’s **subject**.

Returns a unicode string (`TextStringObject`) or `None` if the subject is not specified.

property subject_raw: Optional[str]

The “raw” version of subject; can return a `ByteStringObject`.

property title: Optional[str]

Read-only property accessing the document’s **title**.

Returns a unicode string (`TextStringObject`) or `None` if the title is not specified.

property title_raw: Optional[str]

The “raw” version of title; can return a `ByteStringObject`.

THE XMPINFORMATION CLASS

class PyPDF2.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 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.

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 keywords 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.

getElement(*aboutUri: str, namespace: str, name: str*) → `Iterator[Any]`

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

getNodesInNamespace(*aboutUri: str, namespace: str*) → `Iterator[Any]`

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

get_element(*about_uri: str, namespace: str, name: str*) → `Iterator[Any]`

get_nodes_in_namespace(*about_uri: str, namespace: str*) → `Iterator[Any]`

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 rdfRoot: `Element`

writeToStream(*stream: Union[BytesIO, BufferedReader, BufferedWriter, FileIO], encryption_key: Union[None, str, bytes]*) → `None`

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

write_to_stream(*stream: Union[BytesIO, BufferedReader, BufferedWriter, FileIO], encryption_key: Union[None, str, bytes]*) → `None`

property xmp_createDate: `datetime`

property xmp_create_date: `Optional[Any]`

The date and time the resource was originally created. The date and time are returned as a UTC date-time.datetime object.

property xmp_creatorTool: `str`

property xmp_creator_tool: `Optional[Any]`

The name of the first known tool used to create the resource.

property xmp_metadataDate: `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_modifyDate: 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 xmpmm_documentId: str

property xmpmm_document_id: Optional[Any]

The common identifier for all versions and renditions of this resource.

property xmpmm_instanceId: str

property xmpmm_instance_id: Optional[Any]

An identifier for a specific incarnation of a document, updated each time a file is saved.

THE DESTINATION CLASS

class PyPDF2.generic.Destination(*title: str, page: Union[NumberObject, IndirectObject, NullObject, DictionaryObject], typ: Union[str, NumberObject], *args: Any*)

Bases: TreeObject

A class representing a destination within a PDF file. See section 8.2.1 of the PDF 1.6 reference.

Parameters

- **title** (*str*) – Title of this destination.
- **page** (*IndirectObject*) – Reference to the page of this destination. Should be an instance of *IndirectObject*.
- **typ** (*str*) – How the destination is displayed.
- **args** – Additional arguments may be necessary depending on the type.

Raises

PdfReadError – If destination type is invalid.

Table 1: Valid **typ** arguments (see PDF spec for details)

/Fit	No additional arguments
/XYZ	[left] [top] [zoomFactor]
/FitH	[top]
/FitV	[left]
/FitR	[left] [bottom] [right] [top]
/FitB	No additional arguments
/FitBH	[top]
/FitBV	[left]

property bottom: **Optional[FloatObject]**

Read-only property accessing the bottom vertical coordinate.

property color: **Optional[ArrayObject]**

Read-only property accessing the color in (R, G, B) with values 0.0-1.0

property dest_array: **ArrayObject**

property font_format: **Optional[OutlineFontFlag]**

Read-only property accessing the font type. 1=italic, 2=bold, 3=both

getDestArray() → **ArrayObject**

Deprecated since version 1.28.3: Use *dest_array* instead.

property left: `Optional[FloatObject]`

Read-only property accessing the left horizontal coordinate.

property outline_count: `Optional[int]`

Read-only property accessing the outline count. positive = expanded negative = collapsed absolute value = number of visible descendents at all levels

property page: `Optional[int]`

Read-only property accessing the destination page number.

property right: `Optional[FloatObject]`

Read-only property accessing the right horizontal coordinate.

property title: `Optional[str]`

Read-only property accessing the destination title.

property top: `Optional[FloatObject]`

Read-only property accessing the top vertical coordinate.

property typ: `Optional[str]`

Read-only property accessing the destination type.

write_to_stream(*stream: Union[BytesIO, BufferedReader, BufferedWriter, FileIO], encryption_key: Union[None, str, bytes]*) \rightarrow None

property zoom: `Optional[int]`

Read-only property accessing the zoom factor.

THE RECTANGLEOBJECT CLASS

```
class PyPDF2.generic.RectangleObject(arr: Union[RectangleObject, Tuple[float, float, float, float]])
```

Bases: ArrayObject

This class is used to represent *page boxes* in PyPDF2. These boxes include:

- *artbox*
- *bleedbox*
- *cropbox*
- *mediabox*
- *trimbox*

property bottom: FloatObject

ensureIsNumber(value: Any) → Union[FloatObject, NumberObject]

getHeight() → Decimal

getLowerLeft() → Tuple[Decimal, Decimal]

getLowerLeft_x() → FloatObject

getLowerLeft_y() → FloatObject

getLowerRight() → Tuple[Decimal, Decimal]

getLowerRight_x() → FloatObject

getLowerRight_y() → FloatObject

getUpperLeft() → Tuple[Decimal, Decimal]

getUpperLeft_x() → FloatObject

getUpperLeft_y() → FloatObject

getUpperRight() → Tuple[Decimal, Decimal]

getUpperRight_x() → FloatObject

getUpperRight_y() → FloatObject

getWidth() → Decimal

property height: Decimal

property left: FloatObject

property lowerLeft: Tuple[Decimal, Decimal]

property lowerRight: Tuple[Decimal, Decimal]

property lower_left: Tuple[Decimal, Decimal]

Property to read and modify the lower left coordinate of this box in (x,y) form.

property lower_right: Tuple[Decimal, Decimal]

Property to read and modify the lower right coordinate of this box in (x,y) form.

property right: FloatObject

scale(sx: float, sy: float) → *RectangleObject*

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 top: FloatObject

property upperLeft: Tuple[Decimal, Decimal]

property upperRight: Tuple[Decimal, Decimal]

property upper_left: Tuple[Decimal, Decimal]

Property to read and modify the upper left coordinate of this box in (x,y) form.

property upper_right: Tuple[Decimal, Decimal]

Property to read and modify the upper right coordinate of this box in (x,y) form.

property width: Decimal

THE FIELD CLASS

class PyPDF2.generic.**Field**(*data: Dict[str, Any]*)

Bases: TreeObject

A class representing a field dictionary.

This class is accessed through *get_fields()*

property **additionalActions**: Optional[DictionaryObject]

Deprecated since version 1.28.3.

Use *additional_actions* instead.

property **additional_actions**: Optional[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 **altName**: Optional[str]

Deprecated since version 1.28.3.

Use *alternate_name* instead.

property **alternate_name**: Optional[str]

Read-only property accessing the alternate name of this field.

property **defaultValue**: Optional[Any]

Deprecated since version 1.28.3.

Use *default_value* instead.

property **default_value**: Optional[Any]

Read-only property accessing the default value of this field.

property **fieldType**: Optional[NameObject]

Deprecated since version 1.28.3.

Use *field_type* instead.

property **field_type**: Optional[NameObject]

Read-only property accessing the type of this field.

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 **kids**: Optional[ArrayObject]

Read-only property accessing the kids of this field.

property mappingName: Optional[str]

Deprecated since version 1.28.3.

Use *mapping_name* instead.

property mapping_name: Optional[str]

Read-only property accessing the mapping name of this field. This name is used by PyPDF2 as a key in the dictionary returned by *get_fields()*

property name: Optional[str]

Read-only property accessing the name of this field.

property parent: Optional[DictionaryObject]

Read-only property accessing the parent of this field.

property value: Optional[Any]

Read-only property accessing the value of this field. Format varies based on field type.

THE PAGERANGE CLASS

class PyPDF2.**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).

indices(*n: int*) → Tuple[int, int, int]

n is the length of the list of pages to choose from.

Returns arguments for range(). See help(slice.indices).

to_slice() → slice

Return the slice equivalent of this page range.

static valid(*input: Any*) → bool

True if input is a valid initializer for a PageRange.

THE ANNOTATIONBUILDER CLASS

class PyPDF2.generic.AnnotationBuilder

Bases: object

The AnnotationBuilder creates dictionaries representing PDF annotations.

Those dictionaries can be modified before they are added to a PdfWriter instance via *writer.add_annotation*.

See [adding PDF annotations](#) for it's usage combined with PdfWriter.

```
static free_text(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: str = '000000', background_color: str = 'ffffff') →  
    DictionaryObject
```

Add text in a rectangle to a page.

Parameters

- **text** (*str*) – Text to be added
- **rect** (*RectangleObject*) – or array of four integers specifying the clickable rectangular area [xLL, yLL, xUR, yUR]
- **font** (*str*) – Name of the Font, e.g. 'Helvetica'
- **bold** (*bool*) – Print the text in bold
- **italic** (*bool*) – Print the text in italic
- **font_size** (*str*) – How big the text will be, e.g. '14pt'
- **font_color** (*str*) – Hex-string for the color
- **border_color** (*str*) – Hex-string for the border color
- **background_color** (*str*) – Hex-string for the background of the annotation

```
static line(p1: Tuple[float, float], p2: Tuple[float, float], rect: Union[RectangleObject, Tuple[float, float,  
    float, float]], text: str = "", title_bar: str = "") → DictionaryObject
```

Draw a line on the PDF.

Parameters

- **p1** (*Tuple[float, float]*) – First point
- **p2** (*Tuple[float, float]*) – Second point
- **rect** (*RectangleObject*) – or array of four integers specifying the clickable rectangular area [xLL, yLL, xUR, yUR]
- **text** (*str*) – Text to be displayed as the line annotation

- **title_bar** (*str*) – Text to be displayed in the title bar of the annotation; by convention this is the name of the author

static link(*rect*: ~typing.Union[~PyPDF2.generic._rectangle.RectangleObject, ~typing.Tuple[float, float, float, float]], *border*: ~typing.Optional[~PyPDF2.generic._data_structures.ArrayObject] = None, *url*: ~typing.Optional[str] = None, *target_page_index*: ~typing.Optional[int] = None, *fit*: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *fit_args*: ~typing.Tuple[~typing.Union[~PyPDF2.generic._base.NumberObject, ~PyPDF2.generic._base.NullObject, float], ...] = ()) → DictionaryObject

Add a link to the document.

The link can either be an external link or an internal link.

An external link requires the URL parameter. An internal link requires the target_page_index, fit, and fit args.

Parameters

- **rect** ([RectangleObject](#)) – or array of four integers specifying the clickable rectangular area [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. - horizontal corner radius, - vertical corner radius, and - border width - Optionally: Dash
- **url** (*str*) – Link to a website (if you want to make an external link)
- **target_page_index** (*int*) – index of the page to which the link should go (if you want to make an internal link)
- **fit** (*str*) – Page fit or ‘zoom’ option (see below). Additional arguments may need to be supplied. Passing None will be read as a null value for that coordinate.
- **fit_args** (*Tuple[int, ...]*) – Parameters for the fit argument.

Table 1: Valid fit arguments (see Table 8.2 of the PDF 1.7 reference for details)

/Fit	No additional arguments
/XYZ	[left] [top] [zoomFactor]
/FitH	[top]
/FitV	[left]
/FitR	[left] [bottom] [right] [top]
/FitB	No additional arguments
/FitBH	[top]
/FitBV	[left]

static rectangle(*rect*: Union[[RectangleObject](#), Tuple[float, float, float, float]], *interiour_color*: Optional[str] = None) → DictionaryObject

Draw a rectangle on the PDF.

Parameters

- **rect** ([RectangleObject](#)) – or array of four integers specifying the clickable rectangular area [xLL, yLL, xUR, yUR]

static text(*rect*: Union[[RectangleObject](#), Tuple[float, float, float, float]], *text*: str, *open*: bool = False, *flags*: int = 0) → DictionaryObject

Add text annotation.

Parameters

- **rect**(*Tuple[int, int, int, int]*) – or array of four integers specifying the clickable rectangular area [xLL, yLL, xUR, yUR]
- **open**(*bool*) –
- **flags**(*int*) –

THE PAPERSIZE CLASS

```
class PyPDF2.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)
```


DEVELOPER INTRO

PyPDF2 is a library and hence its users are developers. This document is not for the users, but for people who want to work on PyPDF2 itself.

33.1 Installing Requirements

```
pip install -r requirements/dev.txt
```

33.2 Running Tests

See *testing PyPDF2 with pytest*

33.3 The sample-files git submodule

The reason for having the submodule `sample-files` is that we want to keep the size of the PyPDF2 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
```

33.4 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 PyPDF2 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 PyPDF2, it will automatically be executed when you `git commit`.

33.5 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 PyPDF2.
- **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 PyPDF2 code more consistent. Typically a small change.

33.6 Benchmarks

We need to keep an eye on performance and thus we have a few benchmarks.

See py-pdf.github.io/PyPDF2/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.

34.1 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

34.2 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.

34.3 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
```

34.4 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.

34.5 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.

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
```

35.1 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 (two lowercase f’s).

The two numbers in `begincodespacerange` mean that it starts with an offset of 0 (hence from 1B **FB00**) up 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

PyPDF2 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.

36.1 Semantic Versioning

PyPDF2 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.

36.2 How PyPDF2 deprecates features

Assume the current version of PyPDF2 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 `PendingDeprecationWarning`. 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`: The `PendingDeprecationWarning` is changed to a `DeprecationWarning`. The CHANGELOG informs about it.
3. `(x+2).0.0`: The code and the `DeprecationWarnings` are removed. The CHANGELOG informs about it.

This means the users have 3 warnings in the CHANGELOG, a `PendingDeprecationWarning` until the next major release and a `DeprecationWarning` 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.

TESTING

PyPDF2 uses `pytest` for testing.

37.1 De-selecting groups of tests

PyPDF2 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.
- `external`: 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 external"` or `pytest -m "not samples"`. You can even disable all of them: `pytest -m "not external" -m "not samples" -m "not slow"`.

Please note that this reduces test coverage. The CI will always test all files.

37.2 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.

CHANGELOG

38.1 Version 2.12.0, 2022-12-10

38.1.1 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)

38.1.2 Bug Fixes (BUG)

- Scale PDF annotations (#1479)

38.1.3 Robustness (ROB)

- Padding issue with AES encryption (#1469)
- Accept empty object as null objects (#1477)

38.1.4 Documentation (DOC)

- Add module documentation the PaperSize class (#1447)

38.1.5 Maintenance (MAINT)

- Use 'page_number' instead of 'pagenum' (#1365)
- Add List of pages to PageRangeSpec (#1456)

38.1.6 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](#)

38.2 Version 2.11.2, 2022-11-20

38.2.1 New Features (ENH)

- Add remove_from_tree (#1432)
- Add AnnotationBuilder.rectangle (#1388)

38.2.2 Bug Fixes (BUG)

- JavaScript executed twice (#1439)
- ToUnicode stores /Identity-H instead of stream (#1433)
- Declare Pillow as optional dependency (#1392)

38.2.3 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](#)

38.3 Version 2.11.1, 2022-10-09

38.3.1 Bug Fixes (BUG)

- td matrix (#1373)
- Cope with cmap from #1322 (#1372)

38.3.2 Robustness (ROB)

- Cope with str returned from get_data in cmap (#1380)

[Full Changelog](#)

38.4 Version 2.11.0, 2022-09-25

38.4.1 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)

38.4.2 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)

38.4.3 Robustness (ROB)

- Ensure update_page_form_field_values does not fail if no fields (#1346)

[Full Changelog](#)

38.5 Version 2.10.9, 2022-09-18

38.5.1 New Features (ENH)

- Add rotation property and transfer_rotate_to_content (#1348)

38.5.2 Performance Improvements (PI)

- Avoid string concatenation with large embedded base64-encoded images (#1350)

38.5.3 Bug Fixes (BUG)

- Format floats using their intrinsic decimal precision (#1267)

38.5.4 Robustness (ROB)

- Fix merge_page for pages without resources (#1349)

[Full Changelog](#)

38.6 Version 2.10.8, 2022-09-14

38.6.1 New Features (ENH)

- Add PageObject.user_unit property (#1336)

38.6.2 Robustness (ROB)

- Improve NameObject reading/writing (#1345)

[Full Changelog](#)

38.7 Version 2.10.7, 2022-09-11

38.7.1 Bug Fixes (BUG)

- Fix Error in transformations (#1341)
- Decode #23 in NameObject (#1342)

38.7.2 Testing (TST)

- Use pytest.warns() for warnings, and .raises() for exceptions (#1325)

[Full Changelog](#)

38.8 Version 2.10.6, 2022-09-09

38.8.1 Robustness (ROB)

- Fix infinite loop due to Invalid object (#1331)
- Fix image extraction issue with superfluous whitespaces (#1327)

[Full Changelog](#)

38.9 Version 2.10.5, 2022-09-04

38.9.1 New Features (ENH)

- Process XRefStm (#1297)
- Auto-detect RTL for text extraction (#1309)

38.9.2 Bug Fixes (BUG)

- Avoid scaling cropbox twice (#1314)

38.9.3 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)

38.9.4 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)

38.9.5 Maintenance (MAINT)

- PdfReaderProtocol (#1303)
- Throw PdfReadError if Trailer can't be read (#1298)
- Remove catching OverflowException (#1302)

[Full Changelog](#)

38.10 Version 2.10.4, 2022-08-28

38.10.1 Robustness (ROB)

- Fix errors/warnings on no /Resources within extract_text (#1276)
- Add required line separators in ContentStream ArrayObjects (#1281)

38.10.2 Maintenance (MAINT)

- Use NameObject idempotency (#1290)

38.10.3 Testing (TST)

- Rectangle deletion (#1289)
- Add workflow tests (#1287)
- Remove files after tests ran (#1286)

38.10.4 Packaging (PKG)

- Add minimum version for typing_extensions requirement (#1277)

[Full Changelog](#)

38.11 Version 2.10.3, 2022-08-21

38.11.1 Robustness (ROB)

- Decrypt returns empty bytestring (#1258)

38.11.2 Developer Experience (DEV)

- Modify CI to better verify built package contents (#1244)

38.11.3 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)

38.11.4 Testing (TST)

- Delete annotations (#1263)
- Close PdfMerger in tests (#1260)
- PdfReader.xmp_metadata workflow (#1257)
- Various PdfWriter (Layout, Bookmark deprecation) (#1249)

[Full Changelog](#)

38.12 Version 2.10.2, 2022-08-15

BUG: Add PyPDF2.generic to PyPI distribution

38.13 Version 2.10.1, 2022-08-15

38.13.1 Bug Fixes (BUG)

- TreeObject.remove_child had a non-PdfObject assignment for Count (#1233, #1234)
- Fix stream truncated prematurely (#1223)

38.13.2 Documentation (DOC)

- Fix docstring formatting (#1228)

38.13.3 Maintenance (MAINT)

- Split generic.py (#1229)

38.13.4 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](#)

38.14 Version 2.10.0, 2022-08-07

38.14.1 New Features (ENH)

- “with” support for PdfMerger and PdfWriter (#1193)
- Add AnnotationBuilder.text(...) to build text annotations (#1202)

38.14.2 Bug Fixes (BUG)

- Allow IndirectObjects as stream filters (#1211)

38.14.3 Documentation (DOC)

- Font scrambling
- Page vs Content scaling (#1208)
- Example for orientation parameter of extract_text (#1206)
- Fix AnnotationBuilder parameter formatting (#1204)

38.14.4 Developer Experience (DEV)

- Add flake8-print (#1203)

38.14.5 Maintenance (MAINT)

- Introduce WrongPasswordError / FileNotDecryptedError / EmptyFileError (#1201)

[Full Changelog](#)

38.15 Version 2.9.0, 2022-07-31

38.15.1 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)

38.15.2 Bug Fixes (BUG)

- Named Dest in PDF1.1 (#1174)
- Incomplete Graphic State save/restore (#1172)

38.15.3 Documentation (DOC)

- Update changelog url in package metadata (#1180)
- Mention camelot for table extraction (#1179)
- Mention pyHanko for signing PDF documents (#1178)
- We now have CMAP support since a while (#1177)

38.15.4 Maintenance (MAINT)

- Consistent usage of warnings / log messages (#1164)
- Consistent terminology for outline items (#1156)

[Full Changelog](#)

38.16 Version 2.8.1, 2022-07-25

38.16.1 Bug Fixes (BUG)

- u_hash in AlgV4.compute_key (#1170)

38.16.2 Robustness (ROB)

- Fix loading of file from #134 (#1167)
- Cope with empty DecodeParams (#1165)

38.16.3 Documentation (DOC)

- Typo in merger deprecation warning message (#1166)

38.16.4 Maintenance (MAINT)

- Package updates; solve mypy strict remarks (#1163)

38.16.5 Testing (TST)

- Add test from #325 (#1169)

[Full Changelog](#)

38.17 Version 2.8.0, 2022-07-24

38.17.1 New Features (ENH)

- Add `writer.add_annotation`, `page.annotations`, and `generic.AnnotationBuilder` (#1120)

38.17.2 Bug Fixes (BUG)

- Set `/AS` for `/Btn` form fields in writer (#1161)
- Ignore if `/Perms` verify failed (#1157)

38.17.3 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)

38.17.4 Developer Experience (DEV)

- Introduce `_utils.logger_warning` (#1148)

38.17.5 Maintenance (MAINT)

- Break up `parse_to_unicode` (#1162)
- Add diagnostic output to exception in `read_from_stream` (#1159)
- Reduce `PdfReader.read` complexity (#1151)

38.17.6 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](#)

38.18 Version 2.7.0, 2022-07-21

38.18.1 New Features (ENH)

- Add `outline_count` property (#1129)

38.18.2 Bug Fixes (BUG)

- Make `reader.get_fields` also return dropdowns with options (#1114)
- Add deprecated `EncodedStreamObject` functions back until `PyPDF2==3.0.0` (#1139)

38.18.3 Robustness (ROB)

- Cope with missing `/W` entry (#1136)
- Cope with invalid parent xref (#1133)

38.18.4 Documentation (DOC)

- Contributors file (#1132)
- Fix type in signature of `PdfWriter.add_uri` (#1131)

38.18.5 Developer Experience (DEV)

- Add `.git-blame-ignore-revs` (#1141)

38.18.6 Code Style (STY)

- Fixing typos (#1137)
- Re-use code via `get_outlines_property` in tests (#1130)

[Full Changelog](#)

38.19 Version 2.6.0, 2022-07-17

38.19.1 New Features (ENH)

- Add `color` and `font_format` to `PdfReader.outlines[i]` (#1104)
- Extract Text Enhancement (whitespaces) (#1084)

38.19.2 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)

38.19.3 Documentation (DOC)

- Explanation for git submodule
- Watermark and stamp (#1095)

38.19.4 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)

38.19.5 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)

38.19.6 Code Style (STY)

- Use `IntFlag` for `permissions_flag` / `update_page_form_field_values` (#1094)
- Simplify code (#1101)

[Full Changelog](#)

38.20 Version 2.5.0, 2022-07-10

38.20.1 New Features (ENH)

- Add support for indexed color spaces / `BitsPerComponent` for decoding PNGs (#1067)
- Add `PageObject._get_fonts` (#1083)

38.20.2 Performance Improvements (PI)

- Use iterative DFS in PdfWriter._sweep_indirect_references (#1072)

38.20.3 Bug Fixes (BUG)

- Let Page.scale also scale the crop-/trim-/bleed-/artbox (#1066)
- Column default for CCITTFaxDecode (#1079)

38.20.4 Robustness (ROB)

- Guard against None-value in _get_outlines (#1060)

38.20.5 Documentation (DOC)

- Stamps and watermarks (#1082)
- OCR vs PDF text extraction (#1081)
- Python Version support
- Formatting of CHANGELOG

38.20.6 Developer Experience (DEV)

- Cache downloaded files (#1070)
- Speed-up for CI (#1069)

38.20.7 Maintenance (MAINT)

- Set page.rotate(angle: int) (#1092)
- Issue #416 was fixed by #1015 (#1078)

38.20.8 Testing (TST)

- Image extraction (#1080)
- Image extraction (#1077)

38.20.9 Code Style (STY)

- Apply black
- Typo in Changelog

[Full Changelog](#)

38.21 Version 2.4.2, 2022-07-05

38.21.1 New Features (ENH)

- Add PdfReader.xfa attribute (#1026)

38.21.2 Bug Fixes (BUG)

- Wrong page inserted when PdfMerger.merge is done (#1063)
- Resolve IndirectObject when it refers to a free entry (#1054)

38.21.3 Developer Experience (DEV)

- Added {posargs} to tox.ini (#1055)

38.21.4 Maintenance (MAINT)

- Remove PyPDF2._utils.bytes_type (#1053)

38.21.5 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)

38.21.6 Code Style (STY)

- DOC of compress_content_streams (#1061)
- Minimize diff for #879 (#1049)

[Full Changelog](#)

38.22 Version 2.4.1, 2022-06-30

38.22.1 New Features (ENH)

- Add `writer.pdf_header` property (getter and setter) (#1038)

38.22.2 Performance Improvements (PI)

- Remove `b_call` in `FloatObject.write_to_stream` (#1044)
- Check duplicate objects in `writer._sweep_indirect_references` (#207)

38.22.3 Documentation (DOC)

- How to suppress 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)

38.22.4 Maintenance (MAINT)

- Handle XML error when reading `XmpInformation` (#1030)
- Deduplicate Code / add mutmut config (#1022)

38.22.5 Code Style (STY)

- Use unnecessary one-line function / class attribute (#1043)
- Docstring formatting (#1033)

[Full Changelog](#)

38.23 Version 2.4.0, 2022-06-26

38.23.1 New Features (ENH):

- Support R6 decrypting (#1015)
- Add `PdfReader.pdf_header` (#1013)

38.23.2 Performance Improvements (PI):

- Remove `ord_` calls (#1014)

38.23.3 Bug Fixes (BUG):

- Fix missing page for bookmark (#1016)

38.23.4 Robustness (ROB):

- Deal with invalid Destinations (#1028)

38.23.5 Documentation (DOC):

- `get_form_text_fields` does not extract dropdown data (#1029)
- Adjust `PdfWriter.add_uri` docstring
- Mention `crypto` extra_requires for installation (#1017)

38.23.6 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](#)

38.24 Version 2.3.1, 2022-06-19

BUG: Forgot to add the internal `_codecs` subpackage.

[Full Changelog](#)

38.25 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

38.25.1 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

38.25.2 New Features (ENH)

- Add decrypt support for V5 and AES-128, AES-256 (R5 only) (#749)

38.25.3 Robustness (ROB)

- Fix corrupted (wrongly) linear PDF (#1008)

38.25.4 Maintenance (MAINT)

- Move PDF_Samples folder into resources
- Fix typos (#1007)

38.25.5 Testing (TST)

- Improve encryption/decryption test (#1009)
- Add merger test cases with real PDFs (#1006)
- Add mutmut config

38.25.6 Code Style (STY)

- Put pure data mappings in separate files (#1005)
- Make encryption module private, apply pre-commit (#1010)

[Full Changelog](#)

38.26 Version 2.2.1, 2022-06-17

38.26.1 Performance Improvements (PI)

- Remove b_ calls (#992, #986)
- Apply improvements to _utils suggested by perflint (#993)

38.26.2 Robustness (ROB)

- utf-16-be codec can't decode (...) (#995)

38.26.3 Documentation (DOC)

- Remove reference to Scripts (#987)

38.26.4 Developer Experience (DEV)

- Fix type annotations for add_bookmarks (#1000)

38.26.5 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)

38.26.6 Code Style (STY)

- Apply fixes suggested by pylint (#999)

[Full Changelog](#)

38.27 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](#)

38.28 Version 2.1.1, 2022-06-12

38.28.1 New Features (ENH)

- Add support for pathlib as input for PdfReader (#979)

38.28.2 Performance Improvements (PI)

- Optimize read_next_end_line (#646)

38.28.3 Bug Fixes (BUG)

- Adobe Acrobat ‘Would you like to save this file?’ (#970)

38.28.4 Documentation (DOC)

- Notes on annotations (#982)
- Who uses PyPDF2
- intendet \xe2\x9e\x94 in robustness page (#958)

38.28.5 Maintenance (MAINT)

- pre-commit / requirements.txt updates (#977)
- Mark read_next_end_line as deprecated (#965)
- Export PageObject in PyPDF2 root (#960)

38.28.6 Testing (TST)

- Add MCVE of issue #416 (#980)
- FlateDecode.decode decodeParms (#964)
- Xmp module (#962)
- utils.paeth_predictor (#959)

38.28.7 Code Style (STY)

- Use more tuples and list/dict comprehensions (#976)

[Full Changelog](#)

38.29 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

38.29.1 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 apostrophes 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 l. (#789)
- Update glyphlist (#947) - inspired by #464
- Allow adding PageRange objects (#948)

38.29.2 Bug Fixes (BUG)

- Delete .python-version file (#944)
- Compare StreamObject.decoded_self with None (#931)

38.29.3 Robustness (ROB)

- Fix some conversion errors on non conform PDF (#932)

38.29.4 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)

38.29.5 Developer Experience (DEV)

- Ignore .python-version file
- Mark deprecated code with no-cover (#943)
- Automatically create Github releases from tags (#870)

38.29.6 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)

38.29.7 Code Style (STY)

- black, isort, Flake8, splitting buildCharMap (#950)

[Full Changelog](#)

38.30 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.

38.30.1 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.

38.30.2 New Features (ENH)

- Improve space setting for text extraction (#922)
- Allow setting the decryption password in `PdfReader.__init__` (#920)
- Add `Page.add_transformation` (#883)

38.30.3 Bug Fixes (BUG)

- Fix error adding transformation to page without `/Contents` (#908)

38.30.4 Robustness (ROB)

- Cope with invalid length in streams (#861)

38.30.5 Documentation (DOC)

- Fix style of 1.25 and 1.27 patch notes (#927)
- Transformation (#907)

38.30.6 Developer Experience (DEV)

- Create flake8 config file (#916)
- Use relative imports (#875)

38.30.7 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)

38.30.8 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](#)

38.31 PyPDF2 1.X

See *CHANGELOG PyPDF2 1.X*

CHANGELOG OF PYPDF2 1.X

39.1 Version 1.28.4, 2022-05-29

Bug Fixes (BUG):

- `XmpInformation._converter_date` was unusable (#921)

[Full Changelog](#)

39.2 Version 1.28.3, 2022-05-28

39.2.1 Deprecations (DEP)

- PEP8 renaming (#905)

39.2.2 Bug Fixes (BUG)

- `XmpInformation` missing method `_getText` (#917)
- Fix `PendingDeprecationWarning` on `_merge_page` (#904)

[Full Changelog](#)

39.3 Version 1.28.2, 2022-05-23

39.3.1 Bug Fixes (BUG)

- `PendingDeprecationWarning` for `getContents` (#893)
- `PendingDeprecationWarning` on using `PdfMerger` (#891)

[Full Changelog](#)

39.4 Version 1.28.1, 2022-05-22

39.4.1 Bug Fixes (BUG)

- Incorrectly show deprecation warnings on internal usage (#887)

39.4.2 Maintenance (MAINT)

- Add stacklevel=2 to deprecation warnings (#889)
- Remove duplicate warnings imports (#888)

[Full Changelog](#)

39.5 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](#)

39.5.1 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

39.6 Version 1.27.12, 2022-05-02

39.6.1 Bug Fixes (BUG)

- _rebuild_xref_table expects trailer to be a dict (#857)

39.6.2 Documentation (DOC)

- Security Policy

[Full Changelog](#)

39.7 Version 1.27.11, 2022-05-02

39.7.1 Bug Fixes (BUG)

- Incorrectly issued xref warning/exception (#855)

[Full Changelog](#)

39.8 Version 1.27.10, 2022-05-01

39.8.1 Robustness (ROB)

- Handle missing destinations in reader (#840)
- warn-only in readStringFromStream (#837)
- Fix corruption in startxref or xref table (#788 and #830)

39.8.2 Documentation (DOC)

- Project Governance (#799)
- History of PyPDF2
- PDF feature/version support (#816)
- More details on text parsing issues (#815)

39.8.3 Developer Experience (DEV)

- Add benchmark command to Makefile
- Ignore IronPython parts for code coverage (#826)

39.8.4 Maintenance (MAINT)

- Split pdf module (#836)
- Separated CCITTFax param parsing/decoding (#841)
- Update requirements files

39.8.5 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)

39.8.6 Code Style (STY)

- Apply pre-commit (black, isort) + use snake_case variables (#832)
- Remove debug code (#828)
- Documentation, Variable names (#839)

[Full Changelog](#)

39.9 Version 1.27.9, 2022-04-24

A change I would like to highlight is the performance improvement for large PDF files (#808)

39.9.1 New Features (ENH)

- Add papersizes (#800)
- Allow setting permission flags when encrypting (#803)
- Allow setting form field flags (#802)

39.9.2 Bug Fixes (BUG)

- TypeError in xmp._converter_date (#813)
- Improve spacing for text extraction (#806)
- Fix PDFDocEncoding Character Set (#809)

39.9.3 Robustness (ROB)

- Use null ID when encrypted but no ID given (#812)
- Handle recursion error (#804)

39.9.4 Documentation (DOC)

- CMaps (#811)
- The PDF Format + commit prefixes (#810)
- Add compression example (#792)

39.9.5 Developer Experience (DEV)

- Add Benchmark for Performance Testing (#781)

39.9.6 Maintenance (MAINT)

- Validate PDF magic byte in strict mode (#814)
- Make PdfFileMerger.addBookmark() behave like PdfFileWriters' (#339)
- Quadratic runtime while parsing reduced to linear (#808)

39.9.7 Testing (TST)

- Newlines in text extraction (#807)

[Full Changelog](#)

39.10 Version 1.27.8, 2022-04-21

39.10.1 Bug Fixes (BUG)

- Use 1MB as offset for readNextEndLine (#321)
- 'PdfFileWriter' object has no attribute 'stream' (#787)

39.10.2 Robustness (ROB)

- Invalid float object; use 0 as fallback (#782)

39.10.3 Documentation (DOC)

- Robustness (#785)

[Full Changelog](#)

39.11 Version 1.27.7, 2022-04-19

39.11.1 Bug Fixes (BUG)

- Import exceptions from PyPDF2.errors in PyPDF2.utils (#780)

39.11.2 Code Style (STY)

- Naming in ‘make_changelog.py’

39.12 Version 1.27.6, 2022-04-18

39.12.1 Deprecations (DEP)

- Remove support for Python 2.6 and older (#776)

39.12.2 New Features (ENH)

- Extract document permissions (#320)

39.12.3 Bug Fixes (BUG)

- Clip by trimBox when merging pages, which would otherwise be ignored (#240)
- Add overwriteWarnings parameter PdfFileMerger (#243)
- IndexError for getPage() of decrypted 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)
- Accidental import

39.12.4 Robustness (ROB)

- Allow extra whitespace before “obj” in readObjectHeader (#567)

39.12.5 Documentation (DOC)

- Link to pdftoc in Sample_Code (#628)
- Working with annotations (#764)
- Structure history

39.12.6 Developer Experience (DEV)

- Add issue templates (#765)
- Add tool to generate changelog

39.12.7 Maintenance (MAINT)

- Use grouped constants instead of string literals (#745)
- Add error module (#768)
- Use decorators for @staticmethod (#775)
- Split long functions (#777)

39.12.8 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

39.12.9 Code Style (STY)

- Make variable naming more consistent in tests

[Full changelog](#)

39.13 Version 1.27.5, 2022-04-15

39.13.1 Security (SEC)

- ContentStream_readInlineImage had potential infinite loop (#740)

39.13.2 Bug fixes (BUG)

- Fix merging encrypted files (#757)
- CCITTFaxDecode decodeParms can be an ArrayObject (#756)

39.13.3 Robustness improvements (ROBUST)

- title sometimes None (#744)

39.13.4 Documentation (DOC)

- Adjust short description of the package

39.13.5 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)

39.13.6 Developer Experience Improvements (DEV)

- Remove unused Travis-CI configuration (#747)
- Show code coverage (#754, #755)
- Add mutmut (#760)

39.13.7 Miscellaneous

- STY: Closing file handles, explicit exports, ... (#743)

[Full Changelog](#)

39.14 Version 1.27.4, 2022-04-12

39.14.1 Bug fixes (BUG)

- Guard formatting of `__init__.__doc__` string (#738)

39.14.2 Packaging (PKG)

- Add more precise license field to setup (#733)

39.14.3 Testing (TST)

- Add test for issue #297

39.14.4 Miscellaneous

- DOC: Miscallenious Miscellaneous (Typo)
- TST: Fix CI triggering (master main) (#739)
- STY: Fix various style issues (#742)

[Full Changelog](#)

39.15 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](#)

39.16 Version 1.27.2, 2022-04-09

- Add Scripts (including pdfcats), 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](#)

39.17 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](#)

39.18 Version 1.27.0, 2022-04-07

Features:

- Add alpha channel support for png files in Script (#614)

39.18.1 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)

39.18.2 Documentation (DOC)

- Added Scripts/pdf-image-extractor.py
- Documentation improvements (#550, #538, #324, #426, #394)

39.18.3 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

39.18.4 Developer Experience Improvements (DEV)

- Pre commit: Developers can now `pre-commit install` to avoid tiny issues like trailing whitespaces

39.18.5 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](#)

39.19 Version 1.26.0, 2016-05-18

- NOTE: Active maintenance on PyPDF2 is resuming after a hiatus
- Fixed a bug where image resources were 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)

39.20 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

39.21 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)

39.22 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

39.23 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

39.24 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

39.25 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 egbutler)
- 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)

39.26 Version 1.20, 2014-01-27

- Official Python 3+ support (with contributions from TWAC and cgamman) 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 Hai)
- Exception handling for illegal characters in Name Objects

39.27 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

39.28 Version 1.18, 2013-08-19

- Fixed a bug where older versions 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

39.29 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

39.30 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!

39.31 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.

39.32 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.

39.33 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.

39.34 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.

39.35 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.

39.36 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.

39.37 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.

39.38 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.

39.39 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.

39.40 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.

39.41 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.

39.42 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.

39.43 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.

39.44 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 PyPDF2 project is managed. It describes the different actors, their roles, and the responsibilities they have.

40.1 Terminology

- The **project** is PyPDF2 - 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 technical permissions.
- A **user** is a person who imports PyPDF2 into their code. All PyPDF2 users are developers, but not developers who know the internals of PyPDF2. They only use the public interface of PyPDF2. They will likely have less knowledge about PDF than contributors.
- The **community** is all of that - the users, the contributors, and the maintainers.

40.2 Governance, Leadership, and Steering PyPDF2 forward

PyPDF2 is a free and open source project with over 100 contributors and likely (way) more than 1000 users.

As PyPDF2 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 PyPDF2 forward. During that time, nobody else stepped up to become the new maintainer.

For this reason, PyPDF2 has the **Benevolent Dictator** governance model. The benevolent dictator is a maintainer with all technical permissions - most importantly the permission to push new PyPDF2 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 PyPDF2 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.

40.3 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.

40.4 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 PyPDF2, 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 PyPDF2*.
- 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 appreciate 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. PyPDF2 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.

40.4.1 Issues and Discussions

An issue is any technical description that aims at bringing PyPDF2 forward:

- Bugs tickets: Something went wrong because PyPDF2 developers made a mistake.
- Feature requests: PyPDF2 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: PyPDF2 could be faster - let us know about your specific scenario.

Any comment that is in those technical 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)

40.4.2 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 PyPDF2 maintainers do their best to fix that in a timely manner - please [report such issues](#)!

40.5 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.

HISTORY OF PYPDF2

41.1 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.

41.2 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.

41.3 PyPDF3 and PyPDF4 (2018 - 2022)

Two approaches were made to get PyPDF2 active again: PyPDF3 and PyPDF4.

PyPDF3 had its 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.

41.4 PyPDF2: Reborn (2022-Today)

Martin Thoma took over maintenance of PyPDF2 in April 2022.

CONTRIBUTORS

PyPDF2 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](#).

42.1 Contributors to the pyPdf / PyPDF2 project

- DL6ER
- ediamondscience
- Górny, Michał
- JianzhengLuo
- Karvonen, Harry
- KourFrost
- Lightup1
- Mérino, Antoine
- Pinheiro, Arthur
- programmarchy
- pubpub-zz: involved in community development
- Rogmann, Sascha
- sietzeberends
- Stüber, Timo
- Thoma, Martin: Maintainer of PyPDF2 since April 2022. I hope to build a great community with many awesome contributors. [LinkedIn](#) | [StackOverflow](#) | [Blog](#)
- WevertonGomes
- ztravis

42.2 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 PyPDF2 have the last call on that one.
- Community work: This is exceptional. If the maintainers of PyPDF2 see people being super helpful in answering issues / discussions or being very active on Stackoverflow, we also consider them being contributors to PyPDF2.

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.

PYPDF2 VS X

PyPDF2 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. PyPDF2 can retrieve text and metadata from PDFs as well.

43.1 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 PyPDF2 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.

43.2 pyPDF

PyPDF2 was forked from pyPDF. pyPDF has been unmaintained for a long time.

43.3 PyPDF3 and PyPDF4

Developing and maintaining open source software is extremely time-intensive and in the case of PyPDF2 not paid at all. Having a continuous support is hard.

PyPDF2 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 PyPDF2 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 PyPDF2, hope that we can bring the community back to one path of development. Let's see.

43.4 pdfcrowd and pdfminer

I don't have experience with either of those libraries. Please add a comparison if you know PyPDF2 and [pdfcrowd](#) or [pdfminer.six](#)!

Please be aware that there is also [pdfminer](#) which is not maintained. Then there is [pdfcrowd2](#) which doesn't have a large community behind it.

And there are more:

- [pdfplumber](#)

43.5 Document Generation

There are (Python) [tools to generate PDF documents](#). PyPDF2 is not one of them.

FREQUENTLY-ASKED QUESTIONS

44.1 How is PyPDF2 related to pyPdf?

PyPDF2 is a fork from the no-longer-maintained pyPdf approved by the latter's founder.

44.2 Which Python versions are supported?

PyPDF2 2.0+ supports Python 3.6 and later. PyPDF2 1.27.10 supported Python 2.7 to 3.10.

44.3 Who uses PyPDF2?

pyPdf is vendored into several projects. That means the code of pyPdf was copied into that project.

Projects that depend on PyPDF2:

- [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.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

A

A0 (*PyPDF2.PaperSize* attribute), 113
 A1 (*PyPDF2.PaperSize* attribute), 113
 A2 (*PyPDF2.PaperSize* attribute), 113
 A3 (*PyPDF2.PaperSize* attribute), 113
 A4 (*PyPDF2.PaperSize* attribute), 113
 A5 (*PyPDF2.PaperSize* attribute), 113
 A6 (*PyPDF2.PaperSize* attribute), 113
 A7 (*PyPDF2.PaperSize* attribute), 113
 A8 (*PyPDF2.PaperSize* attribute), 113
 add_annotation() (*PyPDF2.PdfWriter* method), 72
 add_attachment() (*PyPDF2.PdfWriter* method), 72
 add_blank_page() (*PyPDF2.PdfWriter* method), 72
 add_bookmark() (*PyPDF2.PdfMerger* method), 81
 add_bookmark() (*PyPDF2.PdfWriter* method), 72
 add_bookmark_destination() (*PyPDF2.PdfWriter* method), 72
 add_bookmark_dict() (*PyPDF2.PdfWriter* method), 72
 add_js() (*PyPDF2.PdfWriter* method), 72
 add_link() (*PyPDF2.PdfWriter* method), 73
 add_metadata() (*PyPDF2.PdfMerger* method), 81
 add_metadata() (*PyPDF2.PdfWriter* method), 73
 add_named_destination() (*PyPDF2.PdfMerger* method), 82
 add_named_destination() (*PyPDF2.PdfWriter* method), 73
 add_named_destination_object() (*PyPDF2.PdfWriter* method), 73
 add_outline() (*PyPDF2.PdfWriter* method), 73
 add_outline_item() (*PyPDF2.PdfMerger* method), 82
 add_outline_item() (*PyPDF2.PdfWriter* method), 73
 add_outline_item_destination() (*PyPDF2.PdfWriter* method), 73
 add_outline_item_dict() (*PyPDF2.PdfWriter* method), 73
 add_page() (*PyPDF2.PdfWriter* method), 73
 add_transformation() (*PyPDF2._page.PageObject* method), 85
 add_uri() (*PyPDF2.PdfWriter* method), 74
 addAttachment() (*PyPDF2.PdfWriter* method), 71
 addBlankPage() (*PyPDF2.PdfWriter* method), 71

addBookmark() (*PyPDF2.PdfMerger* method), 81
 addBookmark() (*PyPDF2.PdfWriter* method), 71
 addBookmarkDestination() (*PyPDF2.PdfWriter* method), 71
 addBookmarkDict() (*PyPDF2.PdfWriter* method), 71
 additional_actions (*PyPDF2.generic.Field* property), 105
 additionalActions (*PyPDF2.generic.Field* property), 105
 addJS() (*PyPDF2.PdfWriter* method), 71
 addLink() (*PyPDF2.PdfWriter* method), 71
 addMetadata() (*PyPDF2.PdfMerger* method), 81
 addMetadata() (*PyPDF2.PdfWriter* method), 71
 addNamedDestination() (*PyPDF2.PdfMerger* method), 81
 addNamedDestination() (*PyPDF2.PdfWriter* method), 71
 addNamedDestinationObject() (*PyPDF2.PdfWriter* method), 71
 addPage() (*PyPDF2.PdfWriter* method), 72
 addTransformation() (*PyPDF2._page.PageObject* method), 85
 addURI() (*PyPDF2.PdfWriter* method), 72
 alternate_name (*PyPDF2.generic.Field* property), 105
 altName (*PyPDF2.generic.Field* property), 105
 AnnotationBuilder (class in *PyPDF2.generic*), 109
 annotations (*PyPDF2._page.PageObject* property), 85
 append() (*PyPDF2.PdfMerger* method), 82
 append_pages_from_reader() (*PyPDF2.PdfWriter* method), 74
 appendPagesFromReader() (*PyPDF2.PdfWriter* method), 74
 apply_on() (*PyPDF2.Transformation* method), 93
 artBox (*PyPDF2._page.PageObject* property), 85
 artbox (*PyPDF2._page.PageObject* property), 85
 author (*PyPDF2.DocumentInformation* property), 95
 author_raw (*PyPDF2.DocumentInformation* property), 95

B

bleedBox (*PyPDF2._page.PageObject* property), 85
 bleedbox (*PyPDF2._page.PageObject* property), 85

bottom (*PyPDF2.generic.Destination* property), 101
 bottom (*PyPDF2.generic.RectangleObject* property), 103

C

C4 (*PyPDF2.PaperSize* attribute), 113
 cache_get_indirect_object() (*PyPDF2.PdfReader* method), 65
 cache_indirect_object() (*PyPDF2.PdfReader* method), 65
 cacheGetIndirectObject() (*PyPDF2.PdfReader* method), 65
 cacheIndirectObject() (*PyPDF2.PdfReader* method), 65
 clone_document_from_reader() (*PyPDF2.PdfWriter* method), 74
 clone_reader_document_root() (*PyPDF2.PdfWriter* method), 75
 cloneDocumentFromReader() (*PyPDF2.PdfWriter* method), 74
 cloneReaderDocumentRoot() (*PyPDF2.PdfWriter* method), 74
 close() (*PyPDF2.PdfMerger* method), 82
 color (*PyPDF2.generic.Destination* property), 101
 compress() (*PyPDF2.Transformation* static method), 93
 compress_content_streams() (*PyPDF2._page.PageObject* method), 86
 compressContentStreams() (*PyPDF2._page.PageObject* method), 85
 create_blank_page() (*PyPDF2._page.PageObject* static method), 86
 createBlankPage() (*PyPDF2._page.PageObject* static method), 86
 creation_date (*PyPDF2.DocumentInformation* property), 95
 creation_date_raw (*PyPDF2.DocumentInformation* property), 95
 creator (*PyPDF2.DocumentInformation* property), 95
 creator_raw (*PyPDF2.DocumentInformation* property), 95
 cropBox (*PyPDF2._page.PageObject* property), 86
 cropbox (*PyPDF2._page.PageObject* property), 86
 custom_properties (*PyPDF2.xmp.XmpInformation* property), 97

D

dc_contributor (*PyPDF2.xmp.XmpInformation* property), 97
 dc_coverage (*PyPDF2.xmp.XmpInformation* property), 97
 dc_creator (*PyPDF2.xmp.XmpInformation* property), 97
 dc_date (*PyPDF2.xmp.XmpInformation* property), 97

dc_description (*PyPDF2.xmp.XmpInformation* property), 97
 dc_format (*PyPDF2.xmp.XmpInformation* property), 97
 dc_identifier (*PyPDF2.xmp.XmpInformation* property), 97
 dc_language (*PyPDF2.xmp.XmpInformation* property), 97
 dc_publisher (*PyPDF2.xmp.XmpInformation* property), 97
 dc_relation (*PyPDF2.xmp.XmpInformation* property), 97
 dc_rights (*PyPDF2.xmp.XmpInformation* property), 97
 dc_source (*PyPDF2.xmp.XmpInformation* property), 98
 dc_subject (*PyPDF2.xmp.XmpInformation* property), 98
 dc_title (*PyPDF2.xmp.XmpInformation* property), 98
 dc_type (*PyPDF2.xmp.XmpInformation* property), 98
 decode_permissions() (*PyPDF2.PdfReader* method), 65
 decrypt() (*PyPDF2.PdfReader* method), 65
 default_value (*PyPDF2.generic.Field* property), 105
 defaultValue (*PyPDF2.generic.Field* property), 105
 dest_array (*PyPDF2.generic.Destination* property), 101
 Destination (class in *PyPDF2.generic*), 101
 documentInfo (*PyPDF2.PdfReader* property), 65
 DocumentInformation (class in *PyPDF2*), 95

E

encrypt() (*PyPDF2.PdfWriter* method), 75
 ensureIsNumber() (*PyPDF2.generic.RectangleObject* method), 103
 extract_text() (*PyPDF2._page.PageObject* method), 86
 extract_xform_text() (*PyPDF2._page.PageObject* method), 87
 extractText() (*PyPDF2._page.PageObject* method), 86

F

Field (class in *PyPDF2.generic*), 105
 field_type (*PyPDF2.generic.Field* property), 105
 fieldType (*PyPDF2.generic.Field* property), 105
 find_bookmark() (*PyPDF2.PdfMerger* method), 82
 find_outline_item() (*PyPDF2.PdfMerger* method), 83
 flags (*PyPDF2.generic.Field* property), 105
 font_format (*PyPDF2.generic.Destination* property), 101
 free_text() (*PyPDF2.generic.AnnotationBuilder* static method), 109

G

get_contents() (*PyPDF2._page.PageObject* method),

87
 get_destination_page_number() (PyPDF2.PdfReader method), 66
 get_element() (PyPDF2.xmp.XmpInformation method), 98
 get_fields() (PyPDF2.PdfReader method), 66
 get_form_text_fields() (PyPDF2.PdfReader method), 67
 get_named_dest_root() (PyPDF2.PdfWriter method), 75
 get_nodes_in_namespace() (PyPDF2.xmp.XmpInformation method), 98
 get_object() (PyPDF2.PdfReader method), 67
 get_object() (PyPDF2.PdfWriter method), 75
 get_outline_root() (PyPDF2.PdfWriter method), 76
 get_page() (PyPDF2.PdfWriter method), 76
 get_page_number() (PyPDF2.PdfReader method), 67
 get_reference() (PyPDF2.PdfWriter method), 76
 get_threads_root() (PyPDF2.PdfWriter method), 76
 getContents() (PyPDF2._page.PageObject method), 87
 getDestArray() (PyPDF2.generic.Destination method), 101
 getDestinationPageNumber() (PyPDF2.PdfReader method), 66
 getDocumentInfo() (PyPDF2.PdfReader method), 66
 getElement() (PyPDF2.xmp.XmpInformation method), 98
 getFields() (PyPDF2.PdfReader method), 66
 getFormTextFields() (PyPDF2.PdfReader method), 66
 getHeight() (PyPDF2.generic.RectangleObject method), 103
 getIsEncrypted() (PyPDF2.PdfReader method), 66
 getLowerLeft() (PyPDF2.generic.RectangleObject method), 103
 getLowerLeft_x() (PyPDF2.generic.RectangleObject method), 103
 getLowerLeft_y() (PyPDF2.generic.RectangleObject method), 103
 getLowerRight() (PyPDF2.generic.RectangleObject method), 103
 getLowerRight_x() (PyPDF2.generic.RectangleObject method), 103
 getLowerRight_y() (PyPDF2.generic.RectangleObject method), 103
 getNamedDestinations() (PyPDF2.PdfReader method), 66
 getNamedDestRoot() (PyPDF2.PdfWriter method), 75
 getNodesInNamespace() (PyPDF2.xmp.XmpInformation method), 98
 getNumPages() (PyPDF2.PdfReader method), 66
 getNumPages() (PyPDF2.PdfWriter method), 75
 getObject() (PyPDF2.PdfReader method), 66
 getObject() (PyPDF2.PdfWriter method), 75
 getOutlineRoot() (PyPDF2.PdfWriter method), 75
 getOutlines() (PyPDF2.PdfReader method), 66
 getPage() (PyPDF2.PdfReader method), 66
 getPage() (PyPDF2.PdfWriter method), 75
 getPageLayout() (PyPDF2.PdfReader method), 66
 getPageLayout() (PyPDF2.PdfWriter method), 75
 getPageMode() (PyPDF2.PdfReader method), 66
 getPageMode() (PyPDF2.PdfWriter method), 75
 getPageNumber() (PyPDF2.PdfReader method), 66
 getReference() (PyPDF2.PdfWriter method), 75
 getText() (PyPDF2.DocumentInformation method), 95
 getUpperLeft() (PyPDF2.generic.RectangleObject method), 103
 getUpperLeft_x() (PyPDF2.generic.RectangleObject method), 103
 getUpperLeft_y() (PyPDF2.generic.RectangleObject method), 103
 getUpperRight() (PyPDF2.generic.RectangleObject method), 103
 getUpperRight_x() (PyPDF2.generic.RectangleObject method), 103
 getUpperRight_y() (PyPDF2.generic.RectangleObject method), 103
 getWidth() (PyPDF2.generic.RectangleObject method), 103
 getXmpMetadata() (PyPDF2.PdfReader method), 66

H

hash_value_data() (PyPDF2._page.PageObject method), 87
 height (PyPDF2.generic.RectangleObject property), 103

I

images (PyPDF2._page.PageObject property), 87
 indices() (PyPDF2.PageRange method), 107
 insert_blank_page() (PyPDF2.PdfWriter method), 76
 insert_page() (PyPDF2.PdfWriter method), 76
 insertBlankPage() (PyPDF2.PdfWriter method), 76
 insertPage() (PyPDF2.PdfWriter method), 76
 is_encrypted (PyPDF2.PdfReader property), 67
 isEncrypted (PyPDF2.PdfReader property), 67

K

kids (PyPDF2.generic.Field property), 105

L

left (PyPDF2.generic.Destination property), 101
 left (PyPDF2.generic.RectangleObject property), 103

line() (PyPDF2.generic.AnnotationBuilder static method), 109

link() (PyPDF2.generic.AnnotationBuilder static method), 110

lower_left (PyPDF2.generic.RectangleObject property), 104

lower_right (PyPDF2.generic.RectangleObject property), 104

lowerLeft (PyPDF2.generic.RectangleObject property), 104

lowerRight (PyPDF2.generic.RectangleObject property), 104

M

mapping_name (PyPDF2.generic.Field property), 106

mappingName (PyPDF2.generic.Field property), 105

matrix (PyPDF2.Transformation property), 93

mediaBox (PyPDF2._page.PageObject property), 87

mediabox (PyPDF2._page.PageObject property), 88

merge() (PyPDF2.PdfMerger method), 83

merge_page() (PyPDF2._page.PageObject method), 90

mergePage() (PyPDF2._page.PageObject method), 88

mergeRotatedPage() (PyPDF2._page.PageObject method), 88

mergeRotatedScaledPage() (PyPDF2._page.PageObject method), 88

mergeRotatedScaledTranslatedPage() (PyPDF2._page.PageObject method), 88

mergeRotatedTranslatedPage() (PyPDF2._page.PageObject method), 89

mergeScaledPage() (PyPDF2._page.PageObject method), 89

mergeScaledTranslatedPage() (PyPDF2._page.PageObject method), 89

mergeTransformedPage() (PyPDF2._page.PageObject method), 89

mergeTranslatedPage() (PyPDF2._page.PageObject method), 90

metadata (PyPDF2.PdfReader property), 67

modification_date (PyPDF2.DocumentInformation property), 95

modification_date_raw (PyPDF2.DocumentInformation property), 95

N

name (PyPDF2.generic.Field property), 106

named_destinations (PyPDF2.PdfReader property), 67

namedDestinations (PyPDF2.PdfReader property), 67

numPages (PyPDF2.PdfReader property), 67

O

open_destination (PyPDF2.PdfWriter property), 76

outline (PyPDF2.PdfReader property), 67

outline_count (PyPDF2.generic.Destination property), 102

outlines (PyPDF2.PdfReader property), 68

P

page (PyPDF2.generic.Destination property), 102

page_layout (PyPDF2.PdfReader property), 68

page_layout (PyPDF2.PdfWriter property), 77

page_mode (PyPDF2.PdfReader property), 68

page_mode (PyPDF2.PdfWriter property), 77

pageLayout (PyPDF2.PdfReader property), 68

pageLayout (PyPDF2.PdfWriter property), 77

pageMode (PyPDF2.PdfReader property), 68

pageMode (PyPDF2.PdfWriter property), 77

PageObject (class in PyPDF2._page), 85

PageRange (class in PyPDF2), 107

pages (PyPDF2.PdfReader property), 68

pages (PyPDF2.PdfWriter property), 77

PaperSize (class in PyPDF2), 113

parent (PyPDF2.generic.Field property), 106

pdf_header (PyPDF2.PdfReader property), 69

pdf_header (PyPDF2.PdfWriter property), 77

pdf_keywords (PyPDF2.xmp.XmpInformation property), 98

pdf_pdfversion (PyPDF2.xmp.XmpInformation property), 98

pdf_producer (PyPDF2.xmp.XmpInformation property), 98

PdfMerger (class in PyPDF2), 81

PdfReader (class in PyPDF2), 65

PdfWriter (class in PyPDF2), 71

producer (PyPDF2.DocumentInformation property), 96

producer_raw (PyPDF2.DocumentInformation property), 96

R

rdflRoot (PyPDF2.xmp.XmpInformation property), 98

read() (PyPDF2.PdfReader method), 69

read_next_end_line() (PyPDF2.PdfReader method), 69

read_object_header() (PyPDF2.PdfReader method), 69

readNextEndLine() (PyPDF2.PdfReader method), 69

readObjectHeader() (PyPDF2.PdfReader method), 69

rectangle() (PyPDF2.generic.AnnotationBuilder static method), 110

RectangleObject (class in PyPDF2.generic), 103

remove_images() (PyPDF2.PdfWriter method), 78

remove_links() (PyPDF2.PdfWriter method), 78

remove_text() (PyPDF2.PdfWriter method), 78

removeImages() (PyPDF2.PdfWriter method), 77

removeLinks() (PyPDF2.PdfWriter method), 77

removeText() (PyPDF2.PdfWriter method), 78

right (PyPDF2.generic.Destination property), 102
 right (PyPDF2.generic.RectangleObject property), 104
 rotate() (PyPDF2._page.PageObject method), 90
 rotate() (PyPDF2.Transformation method), 93
 rotate_clockwise() (PyPDF2._page.PageObject method), 90
 rotateClockwise() (PyPDF2._page.PageObject method), 90
 rotateCounterClockwise() (PyPDF2._page.PageObject method), 90
 rotation (PyPDF2._page.PageObject property), 91

S

scale() (PyPDF2._page.PageObject method), 91
 scale() (PyPDF2.generic.RectangleObject method), 104
 scale() (PyPDF2.Transformation method), 93
 scale_by() (PyPDF2._page.PageObject method), 91
 scale_to() (PyPDF2._page.PageObject method), 91
 scaleBy() (PyPDF2._page.PageObject method), 91
 scaleTo() (PyPDF2._page.PageObject method), 91
 set_need_appearances_writer() (PyPDF2.PdfWriter method), 78
 set_page_layout() (PyPDF2.PdfMerger method), 83
 set_page_mode() (PyPDF2.PdfMerger method), 83
 set_page_mode() (PyPDF2.PdfWriter method), 78
 setLowerLeft() (PyPDF2.generic.RectangleObject method), 104
 setLowerRight() (PyPDF2.generic.RectangleObject method), 104
 setPageLayout() (PyPDF2.PdfMerger method), 83
 setPageLayout() (PyPDF2.PdfWriter method), 78
 setPageMode() (PyPDF2.PdfMerger method), 83
 setPageMode() (PyPDF2.PdfWriter method), 78
 setUpperLeft() (PyPDF2.generic.RectangleObject method), 104
 setUpperRight() (PyPDF2.generic.RectangleObject method), 104
 subject (PyPDF2.DocumentInformation property), 96
 subject_raw (PyPDF2.DocumentInformation property), 96

T

text() (PyPDF2.generic.AnnotationBuilder static method), 110
 threads (PyPDF2.PdfReader property), 69
 threads (PyPDF2.PdfWriter property), 78
 title (PyPDF2.DocumentInformation property), 96
 title (PyPDF2.generic.Destination property), 102
 title_raw (PyPDF2.DocumentInformation property), 96
 to_slice() (PyPDF2.PageRange method), 107
 top (PyPDF2.generic.Destination property), 102
 top (PyPDF2.generic.RectangleObject property), 104

transfer_rotation_to_content() (PyPDF2._page.PageObject method), 91
 Transformation (class in PyPDF2), 93
 translate() (PyPDF2.Transformation method), 94
 trimBox (PyPDF2._page.PageObject property), 91
 trimbox (PyPDF2._page.PageObject property), 91
 typ (PyPDF2.generic.Destination property), 102

U

update_page_form_field_values() (PyPDF2.PdfWriter method), 78
 updatePageFormFieldValues() (PyPDF2.PdfWriter method), 78
 upper_left (PyPDF2.generic.RectangleObject property), 104
 upper_right (PyPDF2.generic.RectangleObject property), 104
 upperLeft (PyPDF2.generic.RectangleObject property), 104
 upperRight (PyPDF2.generic.RectangleObject property), 104
 user_unit (PyPDF2._page.PageObject property), 91

V

valid() (PyPDF2.PageRange static method), 107
 value (PyPDF2.generic.Field property), 106

W

width (PyPDF2.generic.RectangleObject property), 104
 write() (PyPDF2.PdfMerger method), 84
 write() (PyPDF2.PdfWriter method), 79
 write_stream() (PyPDF2.PdfWriter method), 79
 write_to_stream() (PyPDF2.generic.Destination method), 102
 write_to_stream() (PyPDF2.xmp.XmpInformation method), 98
 writeToStream() (PyPDF2.xmp.XmpInformation method), 98

X

xfa (PyPDF2.PdfReader property), 69
 xmp_create_date (PyPDF2.xmp.XmpInformation property), 98
 xmp_createDate (PyPDF2.xmp.XmpInformation property), 98
 xmp_creator_tool (PyPDF2.xmp.XmpInformation property), 98
 xmp_creatorTool (PyPDF2.xmp.XmpInformation property), 98
 xmp_metadata (PyPDF2.PdfReader property), 69
 xmp_metadata_date (PyPDF2.xmp.XmpInformation property), 98
 xmp_metadataDate (PyPDF2.xmp.XmpInformation property), 98

`xmp_modify_date` (*PyPDF2.xmp.XmpInformation property*), 99
`xmp_modifyDate` (*PyPDF2.xmp.XmpInformation property*), 99
`XmpInformation` (*class in PyPDF2.xmp*), 97
`xmpMetadata` (*PyPDF2.PdfReader property*), 69
`xmppmm_document_id` (*PyPDF2.xmp.XmpInformation property*), 99
`xmppmm_documentId` (*PyPDF2.xmp.XmpInformation property*), 99
`xmppmm_instance_id` (*PyPDF2.xmp.XmpInformation property*), 99
`xmppmm_instanceId` (*PyPDF2.xmp.XmpInformation property*), 99

Z

`zoom` (*PyPDF2.generic.Destination property*), 102