fpdf2 manual

A minimalist PDF creation library for Python

v2.8.7

Table of contents

1. fpdf2

`fpdf2` is a library for simple & fast PDF document generation in Python. It is a fork and the successor of `PyFPDF` (*cf.* history).

Latest Released Version: pypi v2.8.7


```python
from fpdf import FPDF

pdf = FPDF()
pdf.add_page()
pdf.set_font('Helvetica', size=12)
pdf.cell(text="Hello world!")
pdf.output("hello_world.pdf")
```

Go try it **now** online in a Jupyter notebook: Open in Colab or Open In nbviewer

1.1 Main features

- Easy to use, with a user-friendly API, and easy to extend
- Python 3.10+ support
- Unicode (UTF-8) TrueType font subset embedding (Central European, Cyrillic, Greek, Baltic, Thai, Chinese, Japanese, Korean, Hindi and almost any other language in the world)
- Internal / external links

- Embedding images, including transparency and alpha channel, using Pillow (Python Imaging Library)
- Arbitrary path drawing and basic SVG import
- Embedding barcodes, charts & graphs, emojis, symbols & dingbats
- Tables, and also cell / multi-cell / plaintext writing, with automatic page breaks, line break and text justification
- Choice of measurement unit, page format & margins. Optional page header and footer
- Basic conversion from HTML to PDF
- A templating system to render PDFs in batchs
- Images & links alternative descriptions, for accessibility
- Table of contents & document outline
- Document encryption & document signing
- Annotations, including text highlights, and file attachments
- Presentation mode with control over page display duration & transitions
- Optional basic Markdown-like styling: `**bold**, __italics__`
- It has very few dependencies: Pillow, defusedxml, & fonttools
- Can render mathematical equations & charts
- Many example scripts available throughout this documentation, including usage examples with Django, Flask, FastAPI, streamlit, AWS lambdas... : Usage in web APIs
- more than 1300 unit tests with `qpdf`-based PDF diffing, and PDF samples validation using 3 different checkers:

1.2 Tutorials

- English
- Deutsch
- Italian

- español
- français
- □□□□□
- português
- Русский
- Ελληνικά
- עברית
- □□□□
- □□□□□
- □□□□□□□□□
- □□□
- Dutch
- Polski
- Türkçe
- Indonesian
- Slovenščina
- Українська
- □□□□□□

1.3 Installation

From PyPI:

```
pip install fpdf2
```

To get the latest, unreleased, development version straight from the development branch of this repository:

```
pip install git+https://github.com/py-pdf/fpdf2.git@master
```

Development: check the dedicated documentation page.

1.3.1 Displaying deprecation warnings

`DeprecationWarning`s are not displayed by Python by default.

Hence, every time you use a newer version of `fpdf2`, we strongly encourage you to execute your scripts with the `-Wd` option (*cf.* documentation) in order to get warned about deprecated features used in your code.

This can also be enabled programmatically with `warnings.simplefilter('default', DeprecationWarning)`.

1.4 Community

1.4.1 Support

For community support, please feel free to file an issue or open a discussion.

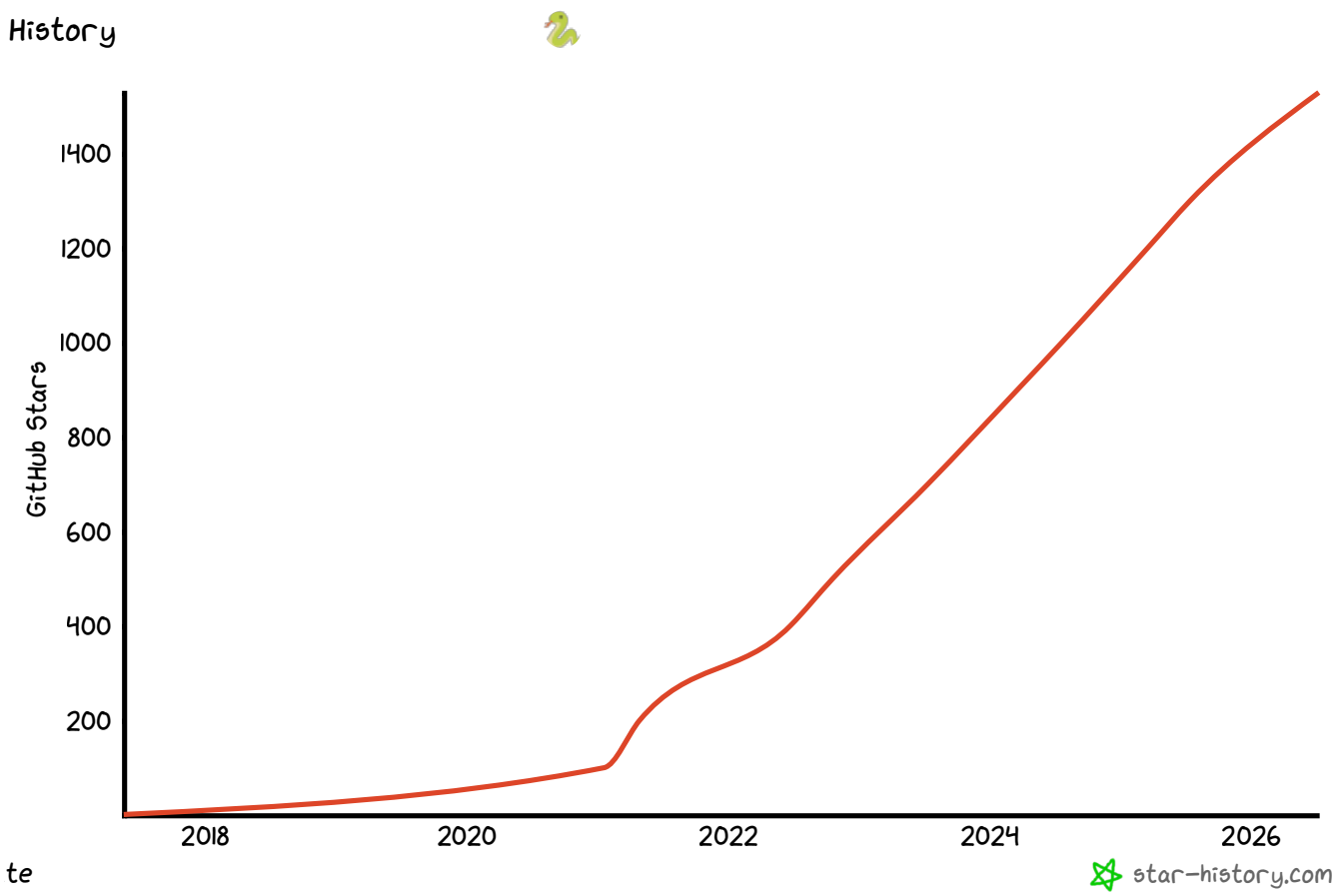
1.4.2 They use fpdf2

Online classes & open source projects:

- **Harvard University** uses `fpdf2` in their CS50 introductory class
- Undying Dusk : a **video game in PDF format**, with a gameplay based on exploration and logic puzzles, in the tradition of dungeon crawlers
- OpenDroneMap : a command line toolkit for processing aerial drone imagery
- OpenSfM : a Structure from Motion library, serving as a processing pipeline for reconstructing camera poses and 3D scenes from multiple images
- RPA Framework : libraries and tools for Robotic Process Automation (RPA), designed to be used with both Robot Framework : rpa-pdf package
- Concordia : a platform developed by the US Library of Congress for crowdsourcing transcription and tagging of text in digitized images
- FreeCAD-Beginner-Assistant : FreeCAD plugin providing feedback on best practices for beginning FreeCAD users
- wudududu/extract-video-ppt : create a one-page-per-frame PDF from a video or PPT file. `fpdf2` also has a demo script to convert a GIF into a one-page-per-frame PDF: gif2pdf.py
- Planet-Matriarchy-RPG-CharGen : a PyQt based desktop application (= `.exe` under Windows) that provides a RPG character sheet generator

1.4.3 Usage statistics

- PyPI download stats - Downloads per release on Pepy
- pip trends: fpdf2 VS other PDF rendering libs
- packages using `fpdf2` can be listed using:
- Wheelodex
- deps.dev
- packages.ecosyste.ms



1.4.4 Related

- Looking for alternative libraries? Check out pypdf, borb, pikepdf, WeasyPrint, pydyf and PyMuPDF: features comparison, examples, Jupyter notebooks. We have some documentations about combining `fpdf2` with `borb` & `pypdf`.
- Create PDFs with Python : a series of tutorial videos by bvalgard
- GitHub gist providing borders around any fpdf2 area, by @hyperstown
- digidigital/Extensions-and-Scripts-for-pyFPDF-fpdf2 : scripts ported from PHP to add transparency to elements of the page or part of an image, allow to write circular text, draw pie charts and bar diagrams, embed JavaScript, draw rectangles with rounded corners, draw a star shape, restrict the rendering of some elements to screen or printout, paint linear / radial / multi-color gradients gradients, add stamps & watermarks, write sheared text...

1.5 Misc

- Release notes for every versions of `fpdf2`: CHANGELOG.md
- This library could only exist thanks to the dedication of many volunteers around the world: list & map of contributors
- You can download an offline PDF version of this manual: fpdf2-manual.pdf

April 10, 2026

2. Tutorial

Open in Colab

Open In nbviewer

2.1 Hello World with fpdf2

This Jupyter notebook demontrates some basic usage of the Python fpdf2 library

```
# Installation of fpdf2 with PIP:
!pip install fpdf2
```

```

```

```
# Enable deprecation warnings:
import warnings
warnings.simplefilter('default', DeprecationWarning)
```

```
# Generate a PDF:
from fpdf import FPDF
pdf = FPDF()
pdf.add_page()
pdf.set_font('helvetica', size=48)
pdf.cell(text="hello world")
pdf_bytes = pdf.output()
```

```
# Display the PDF in the notebook by embedding it as HTML content:
WIDTH, HEIGHT = 800, 400
from base64 import b64encode
from IPython.display import display, HTML
base64_pdf = b64encode(pdf_bytes).decode("utf-8")
display(HTML(f'<embed height="{HEIGHT}" src="data:application/pdf;base64,{base64_pdf}" type="application/pdf" width="{WIDTH}"/>'))
```

```
# Display a download button:
display(HTML(f'<a download="fpdf2-demo.pdf" href="data:application/pdf;base64,{base64_pdf}">Click to download PDF</a>'))
```

Click to download PDF

To continue learning about `fpdf2`, check our tutorial: - English - Deutsch - español - □□□□ - português - Русский - Italian - français - Ελληνικά - עברית - Dutch - Polski - Türkçe - Indonesian

July 10, 2026

2.2 Tutorial

Methods full documentation: `fpdf.FPDF` API doc

2.2.1 Tuto 1 - Minimal Example

Let's start with the classic example:

```
from fpdf import FPDF

pdf = FPDF()
pdf.add_page()
pdf.set_font("helvetica", style="B", size=16)
pdf.cell(40, 10, "Hello World!")
pdf.output("tuto1.pdf")
```

Resulting PDF

After including the library file, we create an `FPDF` object. The FPDF constructor is used here with the default values: pages are in A4 portrait and the measure unit is millimeter. It could have been specified explicitly with:

```
pdf = FPDF(orientation="P", unit="mm", format="A4")
```

It is possible to set the PDF in landscape mode (`L`) or to use other page formats (such as `Letter` and `Legal`) and measure units (`pt` , `cm` , `in`).

There is no page for the moment, so we have to add one with add_page. The origin is at the upper-left corner and the current position is by default placed at 1 cm from the borders; the margins can be changed with set_margins.

Before we can print text, it is mandatory to select a font with set_font, otherwise the document would be invalid. We choose Helvetica bold 16:

```
pdf.set_font('Helvetica', style='B', size=16)
```

We could have specified italics with `I` , underlined with `U` or a regular font with an empty string (or any combination). Note that the font size is given in points, not millimeters (or another user unit); it is the only exception. The other built-in fonts are `Times` , `Courier` , `Symbol` and `ZapfDingbats` .

We can now print a cell with cell. A cell is a rectangular area, possibly framed, which contains some text. It is rendered at the current position. We specify its dimensions, its text (centered or aligned), if borders should be drawn, and where the current position moves after it (to the right, below or to the beginning of the next line). To add a frame, we would do this:

```
pdf.cell(40, 10, 'Hello World!', 1)
```

To add a new cell next to it with centered text and go to the next line, we would do:

```
pdf.cell(60, 10, 'Powered by FPDF.', new_x="LMARGIN", new_y="NEXT", align='C')
```

Remark: the line break can also be done with ln. This method allows to specify in addition the height of the break.

Finally, the document is closed and saved under the provided file path using output. Without any parameter provided, `output()` returns the PDF `bytearray` buffer.

2.2.2 Tuto 2 - Header, footer, page break and image

Here is a two page example with header, footer and logo:

```
from fpdf import FPDF


class PDF(FPDF):
    def header(self):
        # Rendering logo:
        self.image("../docs/fpdf2-logo.png", 10, 8, 33)
```

```
        # Setting font: helvetica bold 15
        self.set_font("helvetica", style="B", size=15)
        # Moving cursor to the right:
        self.cell(80)
        # Printing title:
        self.cell(30, 10, "Title", border=1, align="C")
        # Performing a line break:
        self.ln(20)

    def footer(self):
        # Position cursor at 1.5 cm from bottom:
        self.set_y(-15)
        # Setting font: helvetica italic 8
        self.set_font("helvetica", style="I", size=8)
        # Printing page number:
        self.cell(0, 10, f"Page {self.page_no()}/{{nb}}", align="C")


# Instantiation of inherited class
pdf = PDF()
pdf.add_page()
pdf.set_font("Times", size=12)
for i in range(1, 41):
    pdf.cell(0, 10, f"Printing line number {i}", new_x="LMARGIN", new_y="NEXT")
pdf.output("new-tuto2.pdf")
```

Resulting PDF

This example makes use of the header and footer methods to process page headers and footers. They are called automatically. They already exist in the FPDF class but do nothing, therefore we have to extend the class and override them.

The logo is printed with the image method by specifying its upper-left corner and its width. The height is calculated automatically to respect the image proportions.

To print the page number, a null value is passed as the cell width. It means that the cell should extend up to the right margin of the page; it is handy to center text. The current page number is returned by the page_no method; as for the total number of pages, it is obtained by means of the special value `{nb}` which will be substituted on document closure (this special value can be changed by alias_nb_pages()). Note the use of the set_y method which allows to set position at an absolute location in the page, starting from the top or the bottom.

Another interesting feature is used here: the automatic page breaking. As soon as a cell would cross a limit in the page (at 2 centimeters from the bottom by default), a break is performed and the font restored. Although the header and footer select their own font (`helvetica`), the body continues with `Times`. This mechanism of automatic restoration also applies to colors and line width. The limit which triggers page breaks can be set with set_auto_page_break.

2.2.3 Tuto 3 - Line breaks and colors

Let's continue with an example which prints justified paragraphs. It also illustrates the use of colors.

```
from fpdf import FPDF


class PDF(FPDF):
    def header(self):
        # Setting font: helvetica bold 15
        self.set_font("helvetica", style="B", size=15)
        # Calculating width of title and setting cursor position:
        width = self.get_string_width(self.title) + 6
        self.set_x((210 - width) / 2)
        # Setting colors for frame, background and text:
        self.set_draw_color(0, 80, 180)
        self.set_fill_color(230, 230, 0)
        self.set_text_color(220, 50, 50)
        # Setting thickness of the frame (1 mm)
        self.set_line_width(1)
        # Printing title:
        self.cell(
            width,
            9,
            self.title,
            border=1,
            new_x="LMARGIN",
            new_y="NEXT",
            align="C",
            fill=True,
        )
        # Performing a line break:
        self.ln(10)

    def footer(self):
        # Setting position at 1.5 cm from bottom:
```

```
        self.set_y(-15)
        # Setting font: helvetica italic 8
        self.set_font("helvetica", style="I", size=8)
        # Setting text color to gray:
        self.set_text_color(128)
        # Printing page number
        self.cell(0, 10, f"Page {self.page_no()}", align="C")

    def chapter_title(self, num, label):
        # Setting font: helvetica 12
        self.set_font("helvetica", size=12)
        # Setting background color
        self.set_fill_color(200, 220, 255)
        # Printing chapter name:
        self.cell(
            0,
            6,
            f"Chapter {num} : {label}",
            new_x="LMARGIN",
            new_y="NEXT",
            align="L",
            fill=True,
        )
        # Performing a line break:
        self.ln(4)

    def chapter_body(self, filepath):
        # Reading text file:
        with open(filepath, "rb") as fh:
            txt = fh.read().decode("latin-1")
        # Setting font: Times 12
        self.set_font("Times", size=12)
        # Printing justified text:
        self.multi_cell(0, 5, txt)
        # Performing a line break:
        self.ln()
        # Final mention in italics:
        self.set_font(style="I")
        self.cell(0, 5, "(end of excerpt)")

    def print_chapter(self, num, title, filepath):
        self.add_page()
        self.chapter_title(num, title)
        self.chapter_body(filepath)


pdf = PDF()
pdf.set_title("20000 Leagues Under the Seas")
pdf.set_author("Jules Verne")
pdf.print_chapter(1, "A RUNAWAY REEF", "20k_c1.txt")
pdf.print_chapter(2, "THE PROS AND CONS", "20k_c1.txt")
pdf.output("tuto3.pdf")
```

Resulting PDF

Jules Verne text

The get_string_width method allows determining the length of a string in the current font, which is used here to calculate the position and the width of the frame surrounding the title. Then colors are set (via set_draw_color, set_fill_color and set_text_color) and the thickness of the line is set to 1 mm (against 0.2 by default) with set_line_width. Finally, we output the cell (the last parameter to true indicates that the background must be filled).

The method used to print the paragraphs is multi_cell. Text is justified by default. Each time a line reaches the right extremity of the cell or a carriage return character (`\n`) is met, a line break is issued and a new cell automatically created under the current one. An automatic break is performed at the location of the nearest space or soft-hyphen (`\u00ad`) character before the right limit. A soft-hyphen will be replaced by a normal hyphen when triggering a line break, and ignored otherwise.

Two document properties are defined: the title (set_title) and the author (set_author). Properties can be viewed by two means. First is to open the document directly with Acrobat Reader, go to the File menu and choose the Document Properties option. The second, also available from the plug-in, is to right-click and select Document Properties.

2.2.4 Tuto 4 - Multi Columns

This example is a variant of the previous one, showing how to lay the text across multiple columns.

```
from fpdf import FPDF


class PDF(FPDF):
    def header(self):
        self.set_font("helvetica", style="B", size=15)
```

```
        width = self.get_string_width(self.title) + 6
        self.set_x((210 - width) / 2)
        self.set_draw_color(0, 80, 180)
        self.set_fill_color(230, 230, 0)
        self.set_text_color(220, 50, 50)
        self.set_line_width(1)
        self.cell(
            width,
            9,
            self.title,
            border=1,
            new_x="LMARGIN",
            new_y="NEXT",
            align="C",
            fill=True,
        )
        self.ln(10)

    def footer(self):
        self.set_y(-15)
        self.set_font("helvetica", style="I", size=8)
        self.set_text_color(128)
        self.cell(0, 10, f"Page {self.page_no()}", align="C")

    def chapter_title(self, num, label):
        self.set_font("helvetica", size=12)
        self.set_fill_color(200, 220, 255)
        self.cell(
            0,
            6,
            f"Chapter {num} : {label}",
            new_x="LMARGIN",
            new_y="NEXT",
            border="L",
            fill=True,
        )
        self.ln(4)

    def chapter_body(self, fname):
        # Reading text file:
        with open(fname, "rb") as fh:
            txt = fh.read().decode("latin-1")
        with self.text_columns(
            ncols=3, gutter=5, text_align="J", line_height=1.19
        ) as cols:
            # Setting font: Times 12
            self.set_font("Times", size=12)
            cols.write(txt)
            cols.ln()
            # Final mention in italics:
            self.set_font(style="I")
            cols.write("(end of excerpt)")

    def print_chapter(self, num, title, fname):
        self.add_page()
        self.chapter_title(num, title)
        self.chapter_body(fname)


pdf = PDF()
pdf.set_title("20000 Leagues Under the Seas")
pdf.set_author("Jules Verne")
pdf.print_chapter(1, "A RUNAWAY REEF", "20k_c1.txt")
pdf.print_chapter(2, "THE PROS AND CONS", "20k_c1.txt")
pdf.output("tuto4.pdf")
```

Resulting PDF

Jules Verne text

The key difference from the previous tutorial is the use of the `text_columns` method. It collects all the text, possibly in increments, and distributes it across the requested number of columns, automatically inserting page breaks as necessary. Note that while the `TextColumns` instance is active as a context manager, text styles and other font properties can be changed. Those changes will be contained to the context. Once it is closed the previous settings will be reinstated.

2.2.5 Tuto 5 - Creating Tables

This tutorial will explain how to create two different tables, to demonstrate what can be achieved with some simple adjustments.

```
import csv
from fpdf import FPDF
from fpdf.fonts import FontFace
from fpdf.enums import TableCellFillMode

with open("countries.txt", encoding="utf8") as csv_file:
    data = list(csv.reader(csv_file, delimiter=","))
```

```
pdf = FPDF()
pdf.set_font("helvetica", size=14)

# Basic table:
pdf.add_page()
with pdf.table() as table:
    for data_row in data:
        row = table.row()
        for datum in data_row:
            row.cell(datum)

# Styled table:
pdf.add_page()
pdf.set_draw_color(255, 0, 0)
pdf.set_line_width(0.3)
headings_style = FontFace(emphasis="BOLD", color=255, fill_color=(255, 100, 0))
with pdf.table(
    borders_layout="NO_HORIZONTAL_LINES",
    cell_fill_color=(224, 235, 255),
    cell_fill_mode=TableCellFillMode.ROWS,
    col_widths=(42, 39, 35, 42),
    headings_style=headings_style,
    line_height=6,
    text_align=("LEFT", "CENTER", "RIGHT", "RIGHT"),
    width=160,
) as table:
    for data_row in data:
        row = table.row()
        for datum in data_row:
            row.cell(datum)

pdf.output("tuto5.pdf")
```

Resulting PDF - Countries CSV data

The first example is achieved in the most basic way possible, feeding data to `FPDF.table()`. The result is rudimentary but very quick to obtain.

The second table brings some improvements: colors, limited table width, reduced line height, centered titles, columns with custom widths, figures right aligned... Moreover, horizontal lines have been removed. This was done by picking a `borders_layout` among the available values: `TableBordersLayout`.

2.2.6 Tuto 6 - Creating links and mixing text styles

This tutorial will explain several ways to insert links inside a pdf document, as well as adding links to external sources.

It will also show several ways we can use different text styles, (bold, italic, underline) within the same text.

```
from fpdf import FPDF

pdf = FPDF()

# First page:
pdf.add_page()
pdf.set_font("helvetica", size=20)
pdf.write(5, "To find out what's new in self tutorial, click ")
pdf.set_font(style="U")
link = pdf.add_link(page=2)
pdf.write(5, "here", link)
pdf.set_font()

# Second page:
pdf.add_page()
pdf.image(
    "../docs/fpdf2-logo.png", 10, 10, 50, 0, "", "https://py-pdf.github.io/fpdf2/"
)
pdf.set_left_margin(60)
pdf.set_font_size(18)
pdf.write_html(
    """You can print text mixing different styles using HTML tags: <b>bold</b>, <i>italic</i>,
<u>underlined</u>, or <b><i><u>all at once</u></i></b>!
<br><br>You can also insert links on text, such as <a href="https://py-pdf.github.io/fpdf2/">https://py-pdf.github.io/fpdf2/</a>,
or on an image: the logo is clickable!"""
)
pdf.output("tuto6.pdf")
```

Resulting PDF - fpdf2-logo

The new method shown here to print text is write() . It is very similar to multi_cell() , the key differences being:

- The end of line is at the right margin and the next line begins at the left margin.
- The current position moves to the end of the text.

The method therefore allows us to write a chunk of text, alter the font style, and continue from the exact place we left off. On the other hand, its main drawback is that we cannot justify the text like we do with the multi_cell() method.

In the first page of the example, we used write() for this purpose. The beginning of the sentence is written in regular style text, then using the set_font() method, we switched to underline and finished the sentence.

To add an internal link pointing to the second page, we used the add_link() method, which creates a clickable area which we named "link" that directs to another page within the document.

To create the external link using an image, we used image() . The method has the option to pass a link as one of its arguments. The link can be both internal or external.

As an alternative, another option to change the font style and add links is to use the `write_html()` method. It is an html parser, which allows adding text, changing font style and adding links using html.

2.2.7 Tuto 7 - Creating PDF/A Documents

New in 2.8.3

PDF/A Standards

PDF/A-1 uses PDF-Version 1.4. All resources (pictures, graphics, fonts) must be embedded in the document. The color management must be precise and platform independently specified with ICC-Profiles and the document metadata must be given with XMP-Metadata.

PDF/A-2 uses PDF-Version 1.7. It allows compression with JPEG2000, transparent elements, open type fonts and digital signatures.

The only extension for **PDF/A-3** is the possibility to embed any possible file.

Conformance Classes

Level A (accessible) encompasses all the requirements of the standard, including mapping the content structure and the correct reading order of the document content. Text content must be extractable, and the structure must reflect the natural reading sequence.

Level B (Basic) guarantees a clear visual reproducibility of the content. Level B is generally easier to generate than Level A, but it does not ensure 100 percent text extraction or searchability. The hassle-free reuse of the content is not necessarily given.

To achieve this, here a little example:

```
from pathlib import Path

from fpdf import FPDF
from fpdf import FPDF_VERSION

DIR = Path(__file__).parent
FONT_DIR = DIR / ".." / "test" / "fonts"

pdf = FPDF(enforce_compliance="PDF/A-3B")
pdf.set_lang("en-US")
pdf.set_title("Tutorial7")
pdf.set_author(["John Dow", "Jane Dow"])
pdf.set_subject("Example for PDF/A")
pdf.set_keywords(["example", "tutorial", "fpdf", "pdf/a"])
pdf.set_producer(f"py-pdf/fpdf2 {FPDF_VERSION}")
pdf.add_font(fname=FONT_DIR / "DejaVuSans.ttf")
pdf.add_font("DejaVuSans", style="B", fname=FONT_DIR / "DejaVuSans-Bold.ttf")
pdf.add_font("DejaVuSans", style="I", fname=FONT_DIR / "DejaVuSans-Oblique.ttf")
pdf.add_page()
pdf.set_font("DejaVuSans", style="B", size=20)
pdf.write(text="Header")
pdf.ln(20)
pdf.set_font(size=12)
```

```
pdf.write(text="Example text")
pdf.ln(20)
pdf.set_font(style="I")
pdf.write(text="Example text in italics")

pdf.output("tuto7.pdf")
```

Resulting PDF: tuto7.pdf

Tools like VeraPDF can check conformance of PDF documents produced:

```
verapdf --format text -v tutorial/tuto7.pdf
```

Produces:

```
PASS fpdf2/tutorial/tuto7.pdf 3b
```

September 25, 2025

2.3 Підручник

Повна документація методів: `fpdf.FPDF` – документація API

2.3.1 Туторіал 1 — Мінімальний приклад

Почнемо з класичного прикладу:

```
from fpdf import FPDF

pdf = FPDF()
pdf.add_page()
pdf.set_font("helvetica", style="B", size=16)
pdf.cell(40, 10, "Hello World!")
pdf.output("tuto1.pdf")
```

Отриманий PDF

Після підключення бібліотеки ми створюємо об'єкт `FPDF`. Конструктор FPDF тут використовується зі значеннями за замовчуванням: сторінки формату A4, орієнтація — книжкова (portrait), одиниця виміру — міліметри. Це можна було б задати явно так:

```
pdf = FPDF(orientation="P", unit="mm", format="A4")
```

Можна встановити альбомну орієнтацію сторінки (`L`) або використати інші формати сторінок (наприклад, `Letter` і `Legal`) та одиниці виміру (`pt` , `cm` , `in`).

Поки що сторінок немає, тож треба додати одну за допомогою add_page. Початок координат — у верхньому лівому куті, поточна позиція за замовчуванням розміщена на відстані 1 см від країв; поля можна змінити за допомогою set_margins.

Перш ніж друкувати текст, обов'язково потрібно обрати шрифт за допомогою set_font, інакше документ буде некоректним. Обираємо Helvetica напівжирний розміру 16:

```
pdf.set_font('Helvetica', style='B', size=16)
```

Можна було б вказати курсив через `I` , підкреслення — через `U` , або звичайний шрифт — порожнім рядком (або будь-якою комбінацією). Зверніть увагу, що розмір шрифту задається в пунктах, а не в міліметрах (чи інших одиницях користувача); це єдиний виняток. Інші вбудовані шрифти: `Times` , `Courier` , `Symbol` та `ZapfDingbats` .

Тепер можемо вивести «комірку» за допомогою cell. Комірка — це прямокутна область, за потреби з рамкою, що містить текст. Вона рендериться у поточній позиції. Ми задаємо її розміри, текст (вирівняний по центру або інакше), чи потрібно малювати рамку, а також куди має зміститися поточна позиція після виведення (праворуч, вниз або на початок наступного рядка). Щоб додати рамку, зробимо так:

```
pdf.cell(40, 10, 'Hello World!', 1)
```

Щоб додати поруч нову комірку з текстом, вирівняним по центру, і перейти на новий рядок, зробимо так:

```
pdf.cell(60, 10, 'Powered by FPDF.', new_x="LMARGIN", new_y="NEXT", align='C')
```

Примітка: розрив рядка також можна зробити методом ln. Цей метод додатково дозволяє вказати висоту розриву.

Нарешті, документ закривається й зберігається за вказаним шляхом файлу за допомогою output. Якщо параметри не передано, `output()` повертає PDF як буфер `bytearray` .

2.3.2 Туторіал 2 — Колонтитули, розрив сторінки та зображення

Ось приклад на дві сторінки з верхнім/нижнім колонтитулами та логотипом:

```
from fpdf import FPDF


class PDF(FPDF):
```

```
    def header(self):
        # Rendering logo:
        self.image("../docs/fpdf2-logo.png", 10, 8, 33)
        # Setting font: helvetica bold 15
        self.set_font("helvetica", style="B", size=15)
        # Moving cursor to the right:
        self.cell(80)
        # Printing title:
        self.cell(30, 10, "Title", border=1, align="C")
        # Performing a line break:
        self.ln(20)

    def footer(self):
        # Position cursor at 1.5 cm from bottom:
        self.set_y(-15)
        # Setting font: helvetica italic 8
        self.set_font("helvetica", style="I", size=8)
        # Printing page number:
        self.cell(0, 10, f"Page {self.page_no()}/{{nb}}", align="C")


# Instantiation of inherited class
pdf = PDF()
pdf.add_page()
pdf.set_font("Times", size=12)
for i in range(1, 41):
    pdf.cell(0, 10, f"Printing line number {i}", new_x="LMARGIN", new_y="NEXT")
pdf.output("new-tuto2.pdf")
```

Отриманий PDF

У цьому прикладі використовуються методи header і footer для обробки колонтитулів. Вони викликаються автоматично. Ці методи вже є в класі FPDF, але нічого не роблять, тому ми розширюємо клас і перевизначаємо їх.

Логотип виводиться методом image із зазначенням його координат верхнього лівого кута та ширини. Висота обчислюється автоматично, щоб зберегти пропорції зображення.

Щоб вивести номер сторінки, ширину комірки задаємо нульовою. Це означає, що комірка розширюється до правого поля сторінки; це зручно для центрування тексту. Поточний номер сторінки повертає метод page_no; а загальну кількість сторінок отримують за допомогою спеціального значення `{nb}`, яке буде підставлене під час закриття документа (це значення можна змінити методом alias_nb_pages()). Зверніть увагу на використання методу set_y, який дозволяє встановити позицію в абсолютному місці на сторінці — від верху або віднизу.

Тут також використано цікаву можливість: автоматичні розриви сторінки. Щойно комірка переходить через межу сторінки (стандартно за 2 сантиметри від низу), виконується розрив, а шрифт відновлюється. Хоча в колонтитулах обрано власний шрифт (`helvetica`), основний текст продовжується зі шрифтом `Times`. Цей механізм автоматичного відновлення також стосується кольорів і товщини ліній. Межу, що ініціює розриви сторінок, задають методом set_auto_page_break.

2.3.3 Туторіал 3 — Розриви рядків і кольори

Продовжимо прикладом, що друкує вирівняні по ширині абзаци. Він також демонструє роботу з кольорами.

```
from fpdf import FPDF


class PDF(FPDF):
    def header(self):
        # Setting font: helvetica bold 15
        self.set_font("helvetica", style="B", size=15)
        # Calculating width of title and setting cursor position:
        width = self.get_string_width(self.title) + 6
        self.set_x((210 - width) / 2)
        # Setting colors for frame, background and text:
        self.set_draw_color(0, 80, 180)
        self.set_fill_color(230, 230, 0)
        self.set_text_color(220, 50, 50)
        # Setting thickness of the frame (1 mm)
        self.set_line_width(1)
        # Printing title:
        self.cell(
            width,
            9,
            self.title,
            border=1,
            new_x="LMARGIN",
            new_y="NEXT",
            align="C",
            fill=True,
        )
        # Performing a line break:
        self.ln(10)
```

```
    def footer(self):
        # Setting position at 1.5 cm from bottom:
        self.set_y(-15)
        # Setting font: helvetica italic 8
        self.set_font("helvetica", style="I", size=8)
        # Setting text color to gray:
        self.set_text_color(128)
        # Printing page number
        self.cell(0, 10, f"Page {self.page_no()}", align="C")

    def chapter_title(self, num, label):
        # Setting font: helvetica 12
        self.set_font("helvetica", size=12)
        # Setting background color
        self.set_fill_color(200, 220, 255)
        # Printing chapter name:
        self.cell(
            0,
            6,
            f"Chapter {num} : {label}",
            new_x="LMARGIN",
            new_y="NEXT",
            align="L",
            fill=True,
        )
        # Performing a line break:
        self.ln(4)

    def chapter_body(self, filepath):
        # Reading text file:
        with open(filepath, "rb") as fh:
            txt = fh.read().decode("latin-1")
        # Setting font: Times 12
        self.set_font("Times", size=12)
        # Printing justified text:
        self.multi_cell(0, 5, txt)
        # Performing a line break:
        self.ln()
        # Final mention in italics:
        self.set_font(style="I")
        self.cell(0, 5, "(end of excerpt)")

    def print_chapter(self, num, title, filepath):
        self.add_page()
        self.chapter_title(num, title)
        self.chapter_body(filepath)


pdf = PDF()
pdf.set_title("20000 Leagues Under the Seas")
pdf.set_author("Jules Verne")
pdf.print_chapter(1, "A RUNAWAY REEF", "20k_c1.txt")
pdf.print_chapter(2, "THE PROS AND CONS", "20k_c1.txt")
pdf.output("tuto3.pdf")
```

Отриманий PDF

Текст Жюля Верна

Метод get_string_width дозволяє визначити довжину рядка в поточному шрифті — тут це використовується, щоб розрахувати позицію й ширину рамки навколо заголовка. Далі встановлюємо кольори (через set_draw_color, set_fill_color і set_text_color) та товщину лінії 1 мм (проти 0,2 за замовчуванням) за допомогою set_line_width. Нарешті, виводимо комірку (останній параметр, встановлений у `True`, означає, що тло треба зафарбувати).

Для друку абзаців використовується метод multi_cell. Текст за замовчуванням вирівнюється по ширині. Кожного разу, коли рядок сягає правої межі комірки або зустрічається символ переведення рядка (`\n`), виконується розрив рядка, і автоматично створюється нова комірка під поточною. Автоматичний розрив відбувається в позиції найближчого пробілу або «м'якого переносу» (`\u00ad`) перед правою межею. Під час переносу рядка «м'який перенос» буде замінено на звичайний дефіс; інакше він ігнорується.

Задаються дві властивості документа: заголовок (set_title) та автор (set_author). Переглянути властивості можна двома способами. Перший — відкрити документ у Acrobat Reader, перейти в меню File і обрати Document Properties. Другий — доступний також із плагіна: натиснути правою кнопкою та вибрати Document Properties.

2.3.4 Туторіал 4 — Декілька колонок

Цей приклад — варіант попереднього, демонструє, як розмістити текст у кількох колонках.

```
from fpdf import FPDF
```

```
class PDF(FPDF):
    def header(self):
        self.set_font("helvetica", style="B", size=15)
        width = self.get_string_width(self.title) + 6
        self.set_x((210 - width) / 2)
        self.set_draw_color(0, 80, 180)
        self.set_fill_color(230, 230, 0)
        self.set_text_color(220, 50, 50)
        self.set_line_width(1)
        self.cell(
            width,
            9,
            self.title,
            border=1,
            new_x="LMARGIN",
            new_y="NEXT",
            align="C",
            fill=True,
        )
        self.ln(10)

    def footer(self):
        self.set_y(-15)
        self.set_font("helvetica", style="I", size=8)
        self.set_text_color(128)
        self.cell(0, 10, f"Page {self.page_no()}", align="C")

    def chapter_title(self, num, label):
        self.set_font("helvetica", size=12)
        self.set_fill_color(200, 220, 255)
        self.cell(
            0,
            6,
            f"Chapter {num} : {label}",
            new_x="LMARGIN",
            new_y="NEXT",
            border="L",
            fill=True,
        )
        self.ln(4)

    def chapter_body(self, fname):
        # Reading text file:
        with open(fname, "rb") as fh:
            txt = fh.read().decode("latin-1")
        with self.text_columns(
            ncols=3, gutter=5, text_align="J", line_height=1.19
        ) as cols:
            # Setting font: Times 12
            self.set_font("Times", size=12)
            cols.write(txt)
            cols.ln()
            # Final mention in italics:
            self.set_font(style="I")
            cols.write("(end of excerpt)")

    def print_chapter(self, num, title, fname):
        self.add_page()
        self.chapter_title(num, title)
        self.chapter_body(fname)


pdf = PDF()
pdf.set_title("20000 Leagues Under the Seas")
pdf.set_author("Jules Verne")
pdf.print_chapter(1, "A RUNAWAY REEF", "20k_c1.txt")
pdf.print_chapter(2, "THE PROS AND CONS", "20k_c1.txt")
pdf.output("tuto4.pdf")
```

Отриманий PDF

Текст Жюля Верна

Ключова відмінність від попереднього туторіалу — використання методу `text_columns`. Він збирає весь текст (можливо, по частинах) і розподіляє його на вказану кількість колонок, автоматично вставляючи розриви сторінок за потреби. Зверніть увагу: доки екземпляр `TextColumns` активний як контекстний менеджер, стилі тексту та інші властивості шрифтів можна змінювати — ці зміни будуть обмежені цим контекстом. Після його закриття попередні параметри буде відновлено.

2.3.5 Туторіал 5 — Створення таблиць

У цьому туторіалі пояснюється, як створити дві різні таблиці, щоб продемонструвати, чого можна досягти простими налаштуваннями.

```
import csv
from fpdf import FPDF
from fpdf.fonts import FontFace
from fpdf.enums import TableCellFillMode

with open("countries.txt", encoding="utf8") as csv_file:
    data = list(csv.reader(csv_file, delimiter=","))

pdf = FPDF()
pdf.set_font("helvetica", size=14)

# Basic table:
pdf.add_page()
with pdf.table() as table:
    for data_row in data:
        row = table.row()
        for datum in data_row:
            row.cell(datum)

# Styled table:
pdf.add_page()
pdf.set_draw_color(255, 0, 0)
pdf.set_line_width(0.3)
headings_style = FontFace(emphasis="BOLD", color=255, fill_color=(255, 100, 0))
with pdf.table(
    borders_layout="NO_HORIZONTAL_LINES",
    cell_fill_color=(224, 235, 255),
    cell_fill_mode=TableCellFillMode.ROWS,
    col_widths=(42, 39, 35, 42),
    headings_style=headings_style,
    line_height=6,
    text_align=("LEFT", "CENTER", "RIGHT", "RIGHT"),
    width=160,
) as table:
    for data_row in data:
        row = table.row()
        for datum in data_row:
            row.cell(datum)

pdf.output("tuto5.pdf")
```

Отриманий PDF — CSV-дані про країни

Перший приклад реалізовано максимально простим способом — передаючи дані в `FPDF.table()`. Результат елементарний, але отримується дуже швидко.

Друга таблиця містить покращення: кольори, обмежену ширину таблиці, зменшену висоту рядків, вирівнювання заголовків по центру, колонки з користувацькими ширинами, вирівнювання чисел праворуч... Крім того, горизонтальні лінії прибрано. Цього досягнуто вибором значення `borders_layout` з-поміж доступних у `TableBordersLayout`.

2.3.6 Туторіал 6 — Створення посилань і змішування стилів тексту

У цьому туторіалі показано кілька способів вставки посилань у PDF-документ, а також додавання посилань на зовнішні ресурси.

Також буде показано кілька способів використання різних стилів тексту (напівжирний, курсив, підкреслений) в одному фрагменті.

```
from fpdf import FPDF

pdf = FPDF()

# First page:
pdf.add_page()
pdf.set_font("helvetica", size=20)
pdf.write(5, "To find out what's new in self tutorial, click ")
pdf.set_font(style="U")
link = pdf.add_link(page=2)
pdf.write(5, "here", link)
pdf.set_font()

# Second page:
pdf.add_page()
pdf.image(
    "../docs/fpdf2-logo.png", 10, 10, 50, 0, "", "https://py-pdf.github.io/fpdf2/"
)
pdf.set_left_margin(60)
pdf.set_font_size(18)
pdf.write_html(
    """You can print text mixing different styles using HTML tags: <b>bold</b>, <i>italic</i>,
<u>underlined</u>, or <b><i><u>all at once</u></i></b>!
<br><br>You can also insert links on text, such as <a href="https://py-pdf.github.io/fpdf2/">https://py-pdf.github.io/fpdf2/</a>,
or on an image: the logo is clickable!"""
```

```
)
pdf.output("tuto6.pdf")
```

Отриманий PDF — fpdf2-logo

Новий метод, що використовується тут для виведення тексту, — write(). Він дуже схожий на multi_cell(), але з такими ключовими відмінностями:

- Кінець рядка — на правому полі, а наступний рядок починається від лівого поля.
- Поточна позиція зміщується в кінець виведеного тексту.

Тож метод дозволяє написати фрагмент тексту, змінити стиль шрифту і продовжити з того самого місця. З іншого боку, головний недолік полягає в тому, що не можна вирівняти текст по ширині, як це робить метод multi_cell().

На першій сторінці прикладу ми використали write() саме з цією метою. Початок речення надруковано звичайним стилем, потім за допомогою set_font() перемкнулися на підкреслення і завершили речення.

Щоб додати внутрішнє посилання на другу сторінку, ми використали метод add_link(), який створює клікабельну область (ми назвали її «link»), що веде на іншу сторінку документа.

Щоб створити зовнішнє посилання за допомогою зображення, ми використали image(). Цей метод має параметр для передавання посилання. Посилання може бути як внутрішнім, так і зовнішнім.

Як альтернативу, для зміни стилю шрифту та додавання посилань можна використати метод `write_html()`. Це HTML-розбірник, що дозволяє додавати текст, змінювати стиль шрифту та додавати посилання за допомогою HTML.

2.3.7 Туторіал 7 — Створення документів PDF/A

Нове у *2.8.3*

Стандарти PDF/A

PDF/A-1 використовує PDF-версію 1.4. Усі ресурси (зображення, графіка, шрифти) мають бути вбудовані в документ. Керування кольорами має бути точним і платформонезалежним, із зазначенням профілів ICC, а метадані документа мають бути надані у вигляді XMP-метаданих.

PDF/A-2 використовує PDF-версію 1.7. Дозволяє стиснення JPEG2000, прозорі елементи, шрифти OpenType та цифрові підписи.

Єдине розширення для **PDF/A-3** — можливість вбудовувати будь-які файли.

Класи відповідності

Рівень A (accessible, «доступний») охоплює всі вимоги стандарту, зокрема відображення структурованого змісту та правильний порядок читання вмісту документа. Текст має бути витягуваним, а структура — відображати природну послідовність читання.

Рівень B (Basic, «базовий») гарантує чітке візуальне відтворення вмісту. Рівень B зазвичай легше згенерувати, ніж рівень A, але він не забезпечує 100-відсоткового вилучення тексту чи можливості пошуку. Безпроблемне повторне використання вмісту не обов’язково гарантоване.

Щоб цього досягти, ось невеликий приклад:

```
from pathlib import Path

from fpdf import FPDF
from fpdf import FPDF_VERSION

DIR = Path(__file__).parent
FONT_DIR = DIR / ".." / "test" / "fonts"

pdf = FPDF(enforce_compliance="PDF/A-3B")
pdf.set_lang("en-US")
pdf.set_title("Tutorial7")
pdf.set_author(["John Dow", "Jane Dow"])
pdf.set_subject("Example for PDF/A")
```

```
pdf.set_keywords(["example", "tutorial", "fpdf", "pdf/a"])
pdf.set_producer(f"py-pdf/fpdf2 {FPDF_VERSION}")
pdf.add_font(fname=FONT_DIR / "DejaVuSans.ttf")
pdf.add_font("DejaVuSans", style="B", fname=FONT_DIR / "DejaVuSans-Bold.ttf")
pdf.add_font("DejaVuSans", style="I", fname=FONT_DIR / "DejaVuSans-Oblique.ttf")
pdf.add_page()
pdf.set_font("DejaVuSans", style="B", size=20)
pdf.write(text="Header")
pdf.ln(20)
pdf.set_font(size=12)
pdf.write(text="Example text")
pdf.ln(20)
pdf.set_font(style="I")
pdf.write(text="Example text in italics")

pdf.output("tuto7.pdf")
```

Отриманий PDF: tuto7.pdf

Такі інструменти, як VeraPDF, можуть перевірити відповідність створених PDF-документів:

```
verapdf --format text -v tutorial/tuto7.pdf
```

Результат:

```
PASS fpdf2/tutorial/tuto7.pdf 3b
```

October 22, 2025

3. Page Layout

3.1 Page format and orientation

By default, a `FPDF` document has a `A4` format with `portrait` orientation.

Other formats & orientation can be specified to `FPDF` constructor:

```
pdf = fpdf.FPDF(orientation="landscape", format="A5")
```

Currently supported formats are `a3`, `a4`, `a5`, `letter`, `legal` or a tuple `(width, height)`. Additional standard formats are welcome and can be suggested through pull requests.

3.1.1 Per-page format, orientation and background

`.set_page_background()` lets you set a background for all pages following this call until the background is removed. The value must be of type `str`, `io.BytesIO`, `PIL.Image.Image`, `drawing.DeviceRGB`, `tuple` or `None`

The following code snippet illustrates how to configure different page formats for specific pages as well as setting different backgrounds and then removing it:

```
from fpdf import FPDF

pdf = FPDF()
pdf.set_font("Helvetica")
pdf.set_page_background((252,212,255))
for i in range(9):
    if i == 6:
        pdf.set_page_background('image_path.png')
    pdf.add_page(format=(210 * (1 - i/10), 297 * (1 - i/10)))
    pdf.cell(text=str(i))
pdf.set_page_background(None)
pdf.add_page(same=True)
pdf.cell(text="9")
pdf.output("varying_format.pdf")
```

Similarly, an `orientation` parameter can be provided to the `add_page` method.

3.1.2 Page layout & zoom level

`set_display_mode()` allows to set the **zoom level**: pages can be displayed entirely on screen, occupy the full width of the window, use the real size, be scaled by a specific zooming factor or use the viewer default (configured in its *Preferences* menu).

The **page layout** can also be specified: single page at a time, continuous display, two columns or viewer default.

```
from fpdf import FPDF

pdf = FPDF()
pdf.set_display_mode(zoom="default", layout="TWO_COLUMN_LEFT")
pdf.set_font("helvetica", size=30)
pdf.add_page()
pdf.cell(text="page 1")
pdf.add_page()
pdf.cell(text="page 2")
pdf.output("two-column.pdf")
```

3.1.3 Viewer preferences

Those settings are detailed in the official PDF format specification, but may not be honored by PDF viewers. If a setting seems ignored, this is probably not a bug with `fpdf2`, but a choice or a missing feature from your PDF renderer software.

```
from fpdf import FPDF, ViewerPreferences

pdf = FPDF()
pdf.viewer_preferences = ViewerPreferences(
    hide_toolbar=True,
    hide_menubar=True,
```

```
    hide_window_u_i=True,
    fit_window=True,
    center_window=True,
    display_doc_title=True,
    non_full_screen_page_mode="USE_OUTLINES",
)
pdf.set_font("helvetica", size=30)
pdf.add_page()
pdf.cell(text="page 1")
pdf.add_page()
pdf.cell(text="page 2")
pdf.output("viewer-prefs.pdf")
```

3.1.4 Full screen

```
from fpdf import FPDF

pdf = FPDF()
pdf.page_mode = "FULL_SCREEN"
pdf.output("full-screen.pdf")
```

July 2, 2026

3.2 Margins

By default a `FPDF` document has a 2cm margin at the bottom, and 1cm margin on the other sides.

Those margins control the initial current X & Y position to render elements on a page, and also define the height limit that triggers automatic page breaks when they are enabled.

Margins can be completely removed:

```
pdf.set_margin(0)
```

Several methods can be used to set margins:

- set_margin
- set_left_margin
- set_right_margin
- set_top_margin
- set_margins
- set_auto_page_break

July 2, 2026