

Библиотека Python для создания PDF-документов



cutwoodshop.ru

РЕКЛАМА

3D карта мира из дерева по специальной цене!

Отличный подарок вам и вашим близким. Заказывайте на сайте используя промокод!

Купить

[Сторонние пакеты и модули Python3.](#) / Библиотека Python для создания PDF-документов

[Модуль FPDF2](#) - это библиотека для создания PDF-документов на Python. Представляет собой форк неподдерживаемого модуля PYFPDF.

Основные особенности модуля fpdf2.

- Легко использовать (и легко расширять), получается небольшой и компактный код;
- Не требует установки, компиляции других библиотек (DLL);
- Встраивание подмножества шрифтов Unicode (UTF-8) TrueType;
- Поддержка картинок PNG, GIF и JPG через модуль Pillow (включая прозрачность и альфа-канал);
- Рисование линий и простых форм (круг, квадрат, многоугольник);
- Базовое преобразование из HTML в PDF;
- Поддержка внутренних ссылок, а также внешних ссылок;
- Необязательный базовый стиль, подобный [Markdown](#): **полужирный**, курсив, -подчеркнутый;
- Можно изменять единицу измерения страницы и полей документа;
- Управление верхним и нижним колонтитулами страницы;
- Автоматический разрыв страницы, разрыв строки и выравнивание текста;
- Создание штрих-кода Code 39 и QR-кода;
- Сжатие созданных страниц PDF;
- Пока только одна зависимость: сторонний модуль [Pillow](#).

Установка модуля fpdf2 в виртуальное окружение.

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# ставим модуль fpdf2
(VirtualEnv):~$ python3 -m pip install -U fpdf2
```

Использование русского языка (кириллицы) с модулем fpdf2.

Важно! Модуль fpdf использует latin-1 в качестве кодировки по умолчанию для всех своих встроенных шрифтов, что не позволяет использовать кириллицу (будет выскакивать ошибка `UnicodeEncodeError: 'latin-1' codec can't encode character...`).

Чтобы **создавать PDF-документы на русском языке**, необходимо подключить и использовать TTF-шрифты, поддерживающие более широкий диапазон символов. Например, можно получить бесплатные [шрифты Google Noto](#), которые поддерживают Unicode. Для русского языка, а так же большинства западных языков рекомендуем набор шрифтов NotoSans. Конечно можно использовать и другие шрифты.

Так же можно использовать TTF-шрифты, установленные и используемые операционной системой (подключаются аналогично). Для хранения шрифтов в Linux используются две основных папки: /usr/share/fonts - для глобальных шрифтов и папка ~/.fonts для шрифтов доступных только для пользователя.

Вот как добавить шрифты, поддерживающие кириллицу, и установить их по умолчанию в модуле fpdf2:

```
import fpdf

pdf = fpdf.FPDF()
# подключаем шрифт NotoSans-Regular.ttf и даем ему имя "Sans"
pdf.add_font("Sans", style="", fname="Noto_Sans/NotoSans-Regular.ttf", uni=True)
# подключаем жирный шрифт "Sans"
```

```
pdf.add_font("Sans", style="B", fname="Noto_Sans/NotoSans-Bold.ttf", uni=True)
# подключаем наклонный шрифт "Sans"
pdf.add_font("Sans", style="I", fname="Noto_Sans/NotoSans-Italic.ttf", uni=True)
# подключаем жирный-наклонный шрифт "Sans"
pdf.add_font("Sans", style="BI", fname="Noto_Sans/NotoSans-BoldItalic.ttf", uni=True)
# установить подключенный шрифт "Sans" по умолчанию для всего документа
pdf.set_font("Sans")
```

Основные примеры использования модуля fpdf2 с описанием.

- [Минимальный пример создания PDF-документа](#);
- [Колонтитулы, переносы строк и использование цвета](#);
- [Создание и добавление таблицы в PDF-документ](#);
- [Смешивание стилей текста и создание внутренних ссылок](#).

Минимальный пример создания PDF-документа.

Начнем с минимального примера и разберемся, что там происходит:

```
from fpdf import FPDF

pdf = FPDF()
# включаем TTF шрифты, поддерживающие кириллицу
pdf.add_font("Sans", style="", fname="Noto_Sans/NotoSans-Regular.ttf", uni=True)
pdf.add_font("Sans", style="B", fname="Noto_Sans/NotoSans-Bold.ttf", uni=True)
pdf.add_font("Sans", style="I", fname="Noto_Sans/NotoSans-Italic.ttf", uni=True)
pdf.add_font("Sans", style="BI", fname="Noto_Sans/NotoSans-BoldItalic.ttf", uni=True)
# добавляем пустую страницу
pdf.add_page()
# задаем шрифт `Sans`,
# `Bold` (жирный) и размером 16
pdf.set_font("Sans", "B", 16)

pdf.cell(20, 10, "Добро пожаловать в Python!")
pdf.output("test.pdf")
```

После подключения библиотеки, создается объект FPDF. Здесь используется конструктор FPDF со значениями по умолчанию: формат страницы A4, ориентация портрет, единица измерения - миллиметр.

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

Можно установить ориентацию PDF-документа в альбомный режим (L) или использовать другой формат страниц (например, Letter или A5), а так же изменить единицы измерения (pt, cm, in).

На момент создания экземпляра объекта FPDF страниц нет, поэтому необходимо добавлять их с помощью метода FPDF.add_page(). Начало страницы находится в левом верхнем углу, а текущая позиция по умолчанию располагается на расстоянии 1 см от границ. Поля документа можно изменить с помощью метода [FPDF.set_margins\(\)](#).

Перед добавления текста в PDF-документ, обязательно нужно выбрать шрифт с помощью метода [FPDF.set_font\(\)](#), иначе документ будет недействительным.

```
# задаем шрифт `Sans`, `Bold` (жирный)
# и размером 16 пунктов (не миллиметров!)
pdf.set_font('Sans', 'B', 16)
```

Можно указать курсив с помощью 'I', подчеркнутый шрифт с помощью 'U' или обычный шрифт с помощью пустой строки '' (или использовать любую комбинацию). Обратите внимание, что размер шрифта задается в пунктах, а не в миллиметрах (или другой единице измерений). Это единственное исключение.

Далее создается ячейка с помощью метода FPDF.cell(), которая содержит в себе нужный текст. Ячейка - это прямоугольная область, возможно, обрамленная рамкой, которая содержит некоторый текст. Она отображается в текущей позиции. При ее создании, необходимо указать ее размеры, текст (центрированный или выровненный), должны ли быть нарисованы рамки, и куда перемещается текущая позиция курсора после ее вывода (вправо, вниз или в начало следующей строки).

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

Чтобы добавить новую ячейку с центрированным текстом и перейти к следующей строке, нужно сделать следующее:

```
pdf.cell(60, 10, 'Powered by FPDF.', ln=1, align='C')
```

Примечание: перейти на следующую строку также можно с помощью метода [FPDF.ln\(\)](#). Этот метод позволяет дополнительно указать высоту между строками (высоту разрыва строки).

Наконец, документ закрывается и сохраняется по указанному пути к файлу с помощью метода `FPDF.output()`. Метод `FPDF.output()`, без указания аргументов возвращает буфер PDF [bytearray](#).

Колонтитулы, переносы строк, использование цвета при создании PDF-документа.

Следующий пример печатает выровненные абзацы, а также иллюстрирует использование цвета.

```
from fpdf import FPDF

class PDF(FPDF):

    colontitle = None

    def header(self):
        """Оформление верхнего контитула каждого листа"""
        # Настройка шрифта: Sans, bold, размер 15 пунктов
        self.set_font("Sans", "B", 15)
        # Вычисление ширины заголовка
        # и установка положения курсора
        width = self.get_string_width(self.colontitle) + 6
        self.set_x((210 - width) / 2)
        # Настройка цветов для рамки, фона и текста
        self.set_draw_color(0, 80, 180)
        self.set_fill_color(230, 230, 0)
        self.set_text_color(220, 50, 50)
        # Настройка толщины рамки (1 mm)
        self.set_line_width(1)
        # вывод текста, переданного в `colontitle`
        self.cell(width, 9, self.colontitle, 1, 1, "C", True)
        # Выполнение разрыва строки в 10 мм
        self.ln(10)

    def footer(self):
        """Оформление нижнего контитула каждого листа"""
        # Устанавливаем курсор на 1,5 см от нижнего края
        self.set_y(-15)
        # Настройка шрифта: Sans, italic, 8
        self.set_font("Sans", "I", 8)
        # Установка цвета текста на серый:
        self.set_text_color(128)
        # вывод номера страницы
        self.cell(0, 10, f"Page {self.page_no()}", 0, 0, "C")

    def chapter_title(self, num, label):
        """Оформление главы документа"""
        # Настройка шрифта: Sans 12
        self.set_font("Sans", "", 12)
        # Настройка цвета фона
        self.set_fill_color(200, 220, 255)
        # Печать названия главы
        self.cell(0, 6, f"Глава {num}: {label}", 0, 1, "L", True)
        # Выполнение разрыва строки на 4 мм
        self.ln(4)

    def chapter_body(self, filepath):
        """Чтение файла главы и вывод его в PDF-документ"""
        # Чтение текстового файла:
        with open(filepath, 'r') as fp:
            txt = fp.read()
        # Настройка шрифта: Times, размер 12 пунктов
        self.set_font("Times", size=12)
        # Печать текста:
        self.multi_cell(0, 5, txt)
        # Выполнение разрыва строки:
```

```

self.ln()
# надпись 'Конец главы' выделяем курсивом
self.set_font(style="I")
self.cell(0, 5, "(Конец главы)")

def print_chapter(self, num, title, filepath):
    """Печать одной главы документа"""
    self.add_page()
    self.chapter_title(num, title)
    self.chapter_body(filepath)

pdf = PDF()
# включаем TTF шрифты, поддерживающие кириллицу
pdf.add_font("Sans", style="", fname="Noto_Sans/NotoSans-Regular.ttf", uni=True)
pdf.add_font("Sans", style="B", fname="Noto_Sans/NotoSans-Bold.ttf", uni=True)
pdf.add_font("Sans", style="I", fname="Noto_Sans/NotoSans-Italic.ttf", uni=True)
pdf.add_font("Sans", style="BI", fname="Noto_Sans/NotoSans-BoldItalic.ttf", uni=True)
# Добавляем метаданные только на английском
pdf.set_title("Document title")
pdf.set_author("Author of the document")
# текст верхнего колонтитула документа
pdf.colontitle = "Колонтитул документа"
# Вставляем текст по главам
pdf.print_chapter(1, "Глава 1", "/path/to/chapter1.txt")
pdf.print_chapter(2, "Глава 2", "/path/to/chapter2.txt")
pdf.output("test.pdf")

```

Метод `FPDF.get_string_width()` позволяет определить длину строки в текущем шрифте ('Sans', Bold, 15), которая используется в примере, для расчета положения и ширины рамки, окружающей заголовков. Затем устанавливаются цвета методами `FPDF.set_draw_color()` - цвет линии, `FPDF.set_fill_color()` - цвет заливки фигуры и `FPDF.set_text_color()` - цвет текста. Толщина линии устанавливается в 1 мм (по умолчанию 0,2 мм) с помощью `FPDF.set_line_width()`. Наконец, печатается ячейка (последний параметр `True` указывает на то, что фон должен быть заполнен).

Для печати абзацев используется метод `FPDF.multi_cell()`. Каждый раз, когда строка достигает правого края ячейки или встречается символ возврата каретки, выдается разрыв строки и автоматически создается новая ячейка под текущей. По умолчанию текст выравнивается по ширине.

В примере определены два свойства PDF-документа (метаданные): заголовок `FPDF.set_title()` и автор `FPDF.set_author()`, метаданные можно передавать только в кодировке `latin-1`. Эти метаданные можно просматривать, открыв документ непосредственно с помощью `Acrobat Reader`, перейти в меню "Файл" и выбрать пункт "Свойства документа".

Создание и добавление таблицы в PDF-документ.

В этом примере показывается и рассказывается, как легко можно создавать таблицы в PDF-документе. Нижеприведенный код создает три различные таблицы из 4-х столбцов.

```

from fpdf import FPDF

class PDF(FPDF):
    def basic_table(self, headings, rows):
        for heading in headings:
            self.cell(40, 7, heading, 1)
        self.ln()
        for row in rows:
            for col in row:
                self.cell(40, 6, col, 1)
            self.ln()

    def improved_table(self, headings, rows, col_widths=(42, 39, 35, 40)):
        for col_width, heading in zip(col_widths, headings):
            self.cell(col_width, 7, heading, 1, 0, "C")
        self.ln()
        for row in rows:
            self.cell(col_widths[0], 6, row[0], "LR")
            self.cell(col_widths[1], 6, row[1], "LR")
            self.cell(col_widths[2], 6, row[2], "LR", 0, "R")
            self.cell(col_widths[3], 6, row[3], "LR", 0, "R")
            self.ln()
        # Линия закрытия

```



```

        self.cell(sum(col_widths), 0, "", "T")

def colored_table(self, headings, rows, col_widths=(42, 39, 35, 42)):
    self.set_fill_color(255, 100, 0)
    # цвет текста
    self.set_text_color(255)
    # цвет линий таблицы
    self.set_draw_color(255, 0, 0)
    # ширина линии
    self.set_line_width(0.3)
    # жирный шрифт
    self.set_font(style="B")
    for col_width, heading in zip(col_widths, headings):
        self.cell(col_width, 7, heading, 1, 0, "C", True)
    self.ln()
    # Восстановление цвета и шрифта:
    self.set_fill_color(224, 235, 255)
    self.set_text_color(0)
    self.set_font()
    fill = False
    for row in rows:
        self.cell(col_widths[0], 6, row[0], "LR", 0, "L", fill)
        self.cell(col_widths[1], 6, row[1], "LR", 0, "L", fill)
        self.cell(col_widths[2], 6, row[2], "LR", 0, "R", fill)
        self.cell(col_widths[3], 6, row[3], "LR", 0, "R", fill)
        self.ln()
        fill = not fill
    self.cell(sum(col_widths), 0, "", "T")

def load_data(data, headings=None):
    """Чтение данных из csv"""
    rows = []
    if headings is None:
        headings = []
    for i, row in enumerate(data.splitlines()):
        if not headings and i == 0:
            # извлечение имен столбцов из первой строки:
            headings = row.split(',')
        else:
            rows.append(row.split(','))
    return headings, rows

# Данные, для создания таблицы
data = """Country,Capital,Area (sq km),Population
Algeria,Algiers,2381740,33770000
American Samoa,Pago Pago,199,57500
Andorra,Andorra la Vella,468,72400
Angola,Luanda,1246700,12531000
Anguilla,The Valley,102,14100
Antigua Barbuda,Saint John,443,69800
Argentina,Buenos Aires,2766890,40677000
Armenia,Yerevan,29800,2969000
Aruba,Oranjestad,193,101500
Australia,Canberra,7686850,20601000
Austria,Vienna,83858,8206000
Azerbaijan,Baku,86600,8178000"""

col_names, data = load_data(data)
pdf = PDF()
# включаем TTF шрифты, поддерживающие кириллицу
pdf.add_font("Sans", style="", fname="Noto_Sans/NotoSans-Regular.ttf", uni=True)
pdf.add_font("Sans", style="B", fname="Noto_Sans/NotoSans-Bold.ttf", uni=True)
pdf.add_font("Sans", style="I", fname="Noto_Sans/NotoSans-Italic.ttf", uni=True)
pdf.add_font("Sans", style="BI", fname="Noto_Sans/NotoSans-BoldItalic.ttf", uni=True)
# настройка шрифта
pdf.set_font("Sans", size=14)
pdf.add_page()
# печать таблицы 1
pdf.basic_table(col_names, data)
pdf.add_page()

```

```
# печать таблицы 2
pdf.improved_table(col_names, data)
pdf.add_page()
# печать таблицы 3
pdf.colored_table(col_names, data)
pdf.output("test.pdf")
```

Так как таблица - это просто набор ячеек, то естественно нужно строить таблицу из них.

Пользовательский метод `basic_table()` создает таблицу самым простым способом: простые ячейки в рамке, все одинакового размера и выровненные по левому краю. Результат элементарен, достигается очень быстро.

Пользовательский метод `improved_table()` имеет некоторые улучшения: каждый столбец имеет свою ширину (кортеж `col_widths`), заголовки выровнены по центру, а цифры - по правому краю. Горизонтальные линии не будут прорисовываться. Это достигается при помощи аргумента `border` метода `pdf.cell()`, который указывает, какие стороны ячейки должны быть прорисованы. Здесь нам нужны левая ('L') и правая ('R'). Остается только проблема горизонтальной линии внизу таблицы. Есть две возможности решить ее: проверить наличие последней строки в цикле, в этом случае будем использовать для границы `border='LRB'` или, как сделано в примере, добавим линию после завершения цикла.

Пользовательский метод `improved_table()` создает таблицу, похожую на вторую, но в ней используются цвета. Цвета заливки, текста и линий просто задаются. Альтернативная окраска строк достигается путем использования поочередно прозрачных и заполненных ячеек.

Смешивание стилей текста.

Этот пример показывает несколько способов создания внутренних ссылок PDF-документа, а также добавление ссылок на внешние источники.

Также показывается несколько способов использования различных стилей текста (жирный, курсив, подчеркивание) в одном и том же тексте абзаце/параграфе.

```
from fpdf import FPDF, HTMLMixin

class MyFPDF(FPDF, HTMLMixin):
    pass

pdf = MyFPDF()
# включаем TTF шрифты, поддерживающие кириллицу
pdf.add_font("Sans", style="", fname="Noto_Sans/NotoSans-Regular.ttf", uni=True)
pdf.add_font("Sans", style="B", fname="Noto_Sans/NotoSans-Bold.ttf", uni=True)
pdf.add_font("Sans", style="I", fname="Noto_Sans/NotoSans-Italic.ttf", uni=True)
pdf.add_font("Sans", style="BI", fname="Noto_Sans/NotoSans-BoldItalic.ttf", uni=True)

# Первая страница:
pdf.add_page()
pdf.set_font("Sans", size=20)
pdf.write(5, "Чтобы узнать, что нового в самоучителе, нажмите ")
pdf.set_font(style="U")
# создание внутренней ссылки
link = pdf.add_link()
# добавление внутренней ссылки в текст
pdf.write(5, "здесь", link)
pdf.set_font()

# Вторая страница:
pdf.add_page()
# внутренняя ссылка ведет на элемент
# документа следующий за этим методом
pdf.set_link(link)
pdf.image(
    "/path/to/logo.png", 10, 10, 50, 0, "", "https://docs-python.ru/packages/modul-fpdf2-python/"
)
# задаем отступ слева
pdf.set_left_margin(60)
# размер шрифта
pdf.set_font_size(18)
pdf.write_html(
    """<p>Вы можете печатать текст, смешивая различные стили, используя
теги HTML: <b>bold</b>, <i>italic</i>, <u>underlined</u>,</p>"""
```

```
или <b><i><u>все сразу</u></i></b>!</p>
<p>Вы также можете вставлять ссылки в текст, например
<a href="https://docs-python.ru/packages/modul-fpdf2-python/">Модуль fpdf2 в Python</a>,
или на изображение: картинка `logo.png`, так же кликабельна!</p>""
)
pdf.output("test.pdf")
```

Метод FPDF.write(), это новый метод записи текста в PDF-документ. Он очень похож на FPDF.multi_cell(), основные отличия заключаются в следующем:

- Конец строки находится на правом поле, а следующая строка начинается на левом поле.
- Текущая позиция перемещается в конец текста.

Таким образом, этот метод позволяет писать фрагмент текста, изменить стиль шрифта и продолжить с того самого места, на котором остановились. С другой стороны, его главный недостаток заключается в том, что он не умеет выравнивать текст, как это делает метод FPDF.multi_cell().

На мы использовали для этой цели write(). Начало предложения первой страницы примера записано методом pdf.write(), текстом обычного стиля, а затем, используя метод pdf.set_font(), который переключает шрифт на подчеркивание, далее, следующий методом pdf.write() заканчивает предложение.

Для добавления внутренней ссылки, указывающей на вторую страницу, используется метод pdf.add_link(), создающий кликабельную область, названную 'link', которая ведет в другое место внутри документа. На второй странице используется метод pdf.set_link(), чтобы определить целевую зону для только что созданной ссылки.

Чтобы создать внешнюю ссылку с помощью изображения, используется метод pdf.image(). Этот метод имеет возможность передать ссылку в качестве одного из аргументов. Ссылка может быть как внутренней, так и внешней.

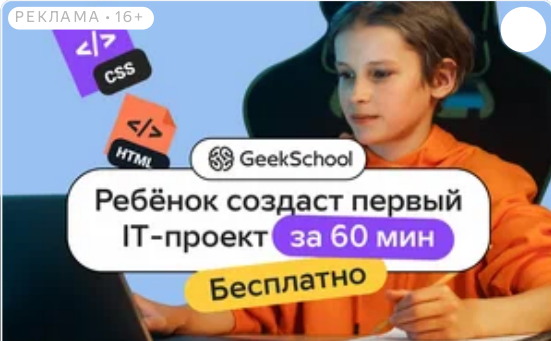
В качестве альтернативы для изменения стиля шрифта и добавления ссылок можно использовать метод pdf.write_html(). Это парсер HTML, который позволяет добавлять текст, изменять стиль шрифта и добавлять ссылки с помощью HTML-тегов.


Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Преобразование HTML в PDF используя модуль fpdf2](#)
- [Определение макета/стиля PDF-документа, модуль fpdf2](#)
- [TTF-шрифты, стилизация текста в PDF, модуль fpdf2](#)
- [Оглавление и схема PDF-документа, модуль fpdf2](#)
- [Добавление таблиц в PDF-документ, модуль fpdf2](#)
- [Добавление картинок в PDF-документ, модуль fpdf2](#)
- [Внешние/внутренние ссылки в PDF, модуль fpdf2](#)
- [Верхний/нижний колонтитулы PDF-документа, модуль fpdf2](#)
- [Печать текста в несколько колонок, модуль fpdf2](#)
- [Отрисовка различных фигур в PDF, модуль fpdf2](#)
- [Вставка математических формул в PDF, модуль fpdf2](#)
- [Создание и добавление штрих/QR-кодов в PDF, модуль fpdf2](#)


ХОЧУ ПОМОЧЬ
ПРОЕКТУ

РЕКЛАМА · 16+





Ребёнок создаст первый
IT-проект **за 60 мин**
Бесплатно

 gb.ru

**Бесплатный
практикум для детей:
Python и анимация**

5,0 ★ Рейтинг организации ⓘ

Модуль fpdf2 в Python, создание PDF-документов

Живой практикум для детей по 2D-анимации и Python. Количество мест ограничено.

Бесплатный мастер-класс >

3 подарка участникам >

Узнать больше