Barcode.Python

Pure Python版 バーコード生成ライブラリ

マニュアル

バージョン 1.0.0

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

Pure Python バーコード生成ライブラリ ユーザーズマニュアル

pip install ひとつで、PDF帳票にも画像にもSVGにも。19種のバーコードを Pure Python で。

バージョン 1.1

2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

1.1 pao_barcodeとは

pao_barcode は、Pure Python で書かれたバーコード生成ライブラリです。

C++で開発された高性能バーコードエンジンを Python に忠実に移植。全19種類のバーコードに対応し、PDF帳票への直接描画、PNG/JPEG画像出力、SVGベクター出力の3つの出力方式をサポートしています。

Pure Python だから、C拡張のコンパイルもNode.jsのインストールも不要。pip install ひとつで、Windows でも macOS でも Linux でも、すぐに使い始められます。

```
# PDF
from reportlab.pdfgen import canvas
from pao_barcode import Code128
from pao_barcode.renderers import ReportLabRenderer

c = canvas.Canvas("hello_barcode.pdf")
renderer = ReportLabRenderer(c)
renderer.draw(Code128(), "Hello-2024", x=50, y=700, width=200, height=50)
c.save()
```

1.2 できること

pao_barcode を使うと、こんなことができます:

やりたいこと	方法	必要なもの
PDF帳票にバーコードを直接描画	ReportLabRenderer	+ reportlab
PNG/JPEG画像ファイルを生成	PillowRenderer	+ Pillow
Base64文字列でWebに表示	PillowRenderer	+ Pillow
SVGベクター画像を生成	エンコーダー内蔵	追加パッケージ不要
既存のPIL画像にバーコードを合成	PillowRenderer.draw()	+ Pillow
バーコードのパターンデータだけ取得	encode() / get_pattern()	追加パッケージ不要

> ヒント: エンコーダーだけなら外部依存ゼロ。SVG出力もエンコーダーに内蔵されているので、pao_barcode単体で動きます。ReportLab や Pillow は、PDF描画や画像出力が必要なときだけ追加すればOKです。

1.3 対応バーコード一覧（19種）

1次元バーコード（11種類）

バーコード	クラス名	どんなところで使われている？
Code39	Code39	工場の部品ラベル、米軍の物資管理
Code93	Code93	郵便局の仕分け、高密度が必要な場面
Code128	Code128	物流ラベル、医療現場、あらゆる業種で
GS1-128	GS1_128	医薬品のトレーサビリティ、コンビニ払込票
NW-7 (Codabar)	NW7	宅配便の送り状、図書館の貸出カード
ITF	ITF	段ボール箱の物流管理（集合包装）
Matrix 2of5	Matrix2of5	航空貨物、工業用途
NEC 2of5	NEC2of5	日本国内の工業用途
JAN-8 (EAN-8)	JAN8	小さなお菓子、リップクリーム
JAN-13 (EAN-13)	JAN13	コンビニ・スーパーの商品コード
UPC-A	UPCA	北米のスーパーマーケットで毎日数十億回

UPCバーコード（1種類）

バーコード	クラス名	どんなところで使われている？
UPC-E	UPCE	小型商品（UPC-Aのゼロ圧縮版）

GS1 DataBar（3種類）

バーコード	クラス名	どんなところで使われている？
GS1 DataBar 標準型	GS1DataBar14	青果・精肉の量り売りラベル
GS1 DataBar 限定型	GS1DataBarLimited	小さな医薬品パッケージ
GS1 DataBar 拡張型	GS1DataBarExpanded	クーポン、有効期限付き商品

2次元バーコード（3種類）

バーコード	クラス名	どんなところで使われている？
QRコード	QR	スマホ決済、Webサイト誘導、名刺
DataMatrix	DataMatrix	電子部品の刻印、手術器具のマーキング
PDF417	PDF417	運転免許証、航空券の搭乗券

特殊バーコード（1種類）

バーコード	クラス名	どんなところで使われている？
郵便カスタマバーコード	YubinCustomer	DMハガキの住所仕分け自動化

> ヒント: 「どのバーコードを使えばいいかわからない」という方へ。迷ったらまず Code128（1D）かQRコード（2D）から始めてみてください。この2つでほとんどのユースケースをカバーできます。

1.4 3つの出力方式

pao_barcode
は「エンコーダー」と「レンダラー」が分離された設計です。エンコーダーがバーコードデータを計算し、レンダラーがそれを可視化します。

出力方式	レンダラー	出力形式	主な用途
PDF帳票	ReportLabRenderer	PDF（ReportLabキャンバスに直接描	納品書、請求書、ラベル印刷
画像	PillowRenderer	PNG / JPEG / BMP / Base64	Webアプリ、画像ファイル
SVG	エンコーダー内蔵	SVGベクター画像	高品質印刷、Web表示

> ヒント: PDF帳票に直接描画できる ReportLabRenderer は、pao_barcodeの最大の強み。「画像をPDFに貼り付ける」のではなく、ベクターデータとして直接描画するため、どんな解像度でもくっきり。しかも、テキスト選択も可能です。

1.5 他のBarcode.Python製品との違い

パオ・アット・オフィスの Python バーコードライブラリには3種類あります:

比較項目	Pure Python版 (ソースコード付) (オ	C++ Import WASM版	C++ Import Native版
パッケージ名	pao_barcode	barcode-pao-wasm	barcode-pao-native
実装	Pure Python	C++ → WebAssembly	C++ → pybind11
環境依存	なし	Node.js必要	C++ビルド環境必要
PDF帳票描画	ReportLabRenderer	--	--
Pillow連携	PillowRenderer	--	--
SVG出力	エンコーダー内蔵	あり	あり
ソースコード	付属	なし	なし
バーコード種類	19種	18種	18種
価格（税込）	22,000円	11,000円	11,000円

Pure Python版を選ぶ理由:

- PDF帳票にバーコードを直接描画したい → ReportLabRenderer は本製品だけの機能
- Pillowで画像処理パイプラインに組み込みたい → PillowRenderer で PIL Image をネイティブに扱える
- 環境を選ばず、どこでも動かしたい → C/C++コンパイラもNode.jsも不要
- ソースコードを確認・カスタマイズしたい → 全ソース付き
- ITFバーコードが必要 → 19種対応は本製品だけ

> ヒント: 「PDF帳票にバーコードを入れたい」なら Pure Python版 (pao_barcode) 一択です。ReportLabのキャンバスに直接描画できるのは、この製品だけ。納品書、請求書、ラベル、コンビニ払込票 — あらゆる業務帳票に対応します。

1.6 デモサイト

実際の動作をブラウザで確認できるデモサイトを公開しています:

- Pure Python版デモ（All-in-One）: https://www.pao.ac/demo/barcode-python/
- PIP版デモ（All-in-One）: https://www.pao.ac/demo/barcode-python-pip/

全19種のバーコード生成、PDF帳票出力、画像出力をお試しいただけます。

2.1 基本インストール

```
pip install pao_barcode
```

これだけです。Pure Python なので、プラットフォーム固有のビルドは不要。

要件:

- Python 3.8 以上
- Windows / macOS / Linux

2.2 オプション依存パッケージ

用途に応じて追加インストールしてください:

ReportLab（PDF帳票出力に必要）

```
pip install reportlab
```

PDF帳票のキャンバスにバーコードを直接描画する ReportLabRenderer を使うために必要です。

Pillow（画像出力に必要）

```
pip install Pillow
```

PNG/JPEG/BMP画像を出力する PillowRenderer を使うために必要です。

全部入りインストール

```
pip install pao_barcode reportlab Pillow
```

> ヒント: エンコーダー（encode() / get_pattern()）とSVG出力だけなら、pao_barcode 単体で動きます。ReportLab や Pillow は必要になったときに追加すれば大丈夫。

2.3 動作確認

```
import pao_barcode
print(pao_barcode.__version__)  # '1.0.0'

#
from pao_barcode import Code128
bc = Code128()
pattern = bc.encode("TEST1234")
print(f"Code128 : {len(pattern)} ")
# → Code128 : 33

# QR
from pao_barcode import QR
qr = QR()
matrix = qr.get_pattern("Hello World")
print(f"QR: {len(matrix)}x{len(matrix[0])} ")
# → QR: 25x25
```

3.1 エンコーダーとレンダラー

pao_barcode は エンコーダー（バーコードデータの計算）とレンダラー（描画）を分離した設計です。エンコーダーが「バーの並び」を計算し、レンダラーがそれを「目に見える画像やPDF」にします。

```
from pao_barcode import Code128, QR, JAN13

# --- 1D ---
# encode() →
bc = Code128()
pattern = bc.encode("Hello-2024")
# pattern = [2, 1, 1, 2, 3, 2, ...]  ← bar, space, bar, ...

# --- 2D ---
# get_pattern() → 2D
qr = QR()
matrix = qr.get_pattern("https://www.pao.ac/")
# matrix[row][col] = True() or False()

# --- JAN/UPC ---
jan = JAN13()
pattern = jan.encode("4901234567894")
```

エンコーダーの出力は数値データだけ。描画には ReportLabRenderer（PDF用）か PillowRenderer（画像用）を使います。

> ヒント: エンコーダーの出力を独自のレンダラーで描画することも可能です。たとえば encode() の結果を使って、自前のCanvasやSVG描画ロジックに組み込めます。

3.2 Pillowで画像出力

```
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()
bc = Code128()

# PIL Image
img = renderer.render(bc, "Hello-2024", width=400, height=100)
img.show()  #

# PNG
renderer.render_to_file(bc, "Hello-2024", "barcode.png", 400, 100)

# Base64 Web
data_uri = renderer.render_to_base64(bc, "Hello-2024", 400, 100)
html = f'<img src="{data_uri}">'
```

3.3 ReportLabでPDF出力

```
from reportlab.pdfgen import canvas
from pao_barcode import Code128
from pao_barcode.renderers import ReportLabRenderer

# ReportLab
c = canvas.Canvas("barcode.pdf")

#
renderer = ReportLabRenderer(c)

# PDF
bc = Code128()
renderer.draw(bc, "Hello-2024", x=50, y=700, width=200, height=50)

# PDF
c.save()
```

> ヒント: ReportLabRenderer は PDF のキャンバスにベクターデータとして直接描画します。画像を貼り付けるのとは違い、拡大してもくっきり、ファイルサイズも小さい。業務帳票にバーコードを入れるなら、これがベストプラクティスです。

3.4 SVGベクター出力（レンダラー不要）

SVG出力はエンコーダー単体で行えます。レンダラーは不要です。

```
from pao_barcode import Code128

bc = Code128()
bc.set_output_format("svg")    # SVG
bc.set_show_text(True)

#  → SVG
bc.draw("Hello-2024", 0, 0, 400, 100)
svg_string = bc.get_svg()

#
with open("barcode.svg", "w", encoding="utf-8") as f:
    f.write(svg_string)
```

> ヒント: SVG出力は追加パッケージ不要。Webページへの埋め込みや、Illustratorでの編集など、ベクターデータが欲しいときに便利です。

pao_barcode の最大の強み、それは ReportLab で作成するPDF帳票にバーコードを直接描画できること。このセクションでは、実際の業務帳票を想定した5つの実践コードを紹介します。

4.1 ReportLabRendererの基本

```
from reportlab.pdfgen import canvas
from pao_barcode import Code128
from pao_barcode.renderers import ReportLabRenderer

# 1. ReportLab
c = canvas.Canvas("output.pdf")

# 2. ReportLabRenderer
renderer = ReportLabRenderer(c)

# 3.
bc = Code128()

# 4. : draw(, , x, y, , )
renderer.draw(bc, "1234567890", x=50, y=700, width=200, height=50)

# 5.
from pao_barcode import QR
qr = QR()
renderer.draw(qr, "https://www.pao.ac/", x=50, y=600, width=80, height=80)

# 6. PDF
c.save()
```

座標系について:

- ReportLab の座標系は 原点が左下、Y軸が上向き です
- (x, y) はバーコード描画領域の 左下隅 を指定します
- 単位は ポイント（1ポイント = 1/72インチ）
- A4用紙は 595 x 842 ポイント

ReportLabRendererのオプション:

```
renderer = ReportLabRenderer(
    c,
    fore_color=(0, 0, 0),        #  (R,G,B) 0-255:
    back_color=(255, 255, 255),   #  (R,G,B) or None():
    show_text=True,               # : True
    font_name='Helvetica',        # : 'Helvetica'
    font_size=8,                  # (pt): 8
    quiet_zone=0.05,              # : 0.05 (5%)
)
```

> ヒント: 1つの ReportLabRenderer インスタンスで、同じページに何種類ものバーコードを描画できます。エンコーダーを切り替えるだけ。

4.2 実例1: 納品書

注文番号のバーコードとQRコードを含む納品書PDFです。Code128で注文番号を、QRコードで問い合わせURLを埋め込みます。

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from pao_barcode import Code128, QR
from pao_barcode.renderers import ReportLabRenderer

def create_delivery_note(file_path, order_no, items):
    """PDF"""
    page_w, page_h = A4
    c = canvas.Canvas(file_path, pagesize=A4)
    renderer = ReportLabRenderer(c, font_size=7)

    # ---  ---
    c.setFont("Helvetica-Bold", 18)
    c.drawCentredString(page_w / 2, page_h - 40, "  ")

    c.setFont("Helvetica", 10)
    c.drawString(50, page_h - 70, f": {order_no}")
    c.drawString(50, page_h - 85, ": 2026-02-05")

    # --- ---
    bc = Code128()
    renderer.draw(bc, order_no,
                  x=page_w - 220, y=page_h - 100,
                  width=180, height=45)

    # ---  ---
    y = page_h - 130
    c.setFont("Helvetica-Bold", 9)
    c.drawString(50, y, "")
    c.drawString(200, y, "")
    c.drawString(400, y, "")
    c.drawString(470, y, "")
    c.line(50, y - 5, 545, y - 5)

    c.setFont("Helvetica", 9)
    y -= 20
    total = 0
    for code, name, qty, price in items:
        c.drawString(50, y, code)
        c.drawString(200, y, name)
        c.drawString(400, y, str(qty))
        c.drawString(470, y, f"¥{price:,}")
        total += qty * price
```

```
            y -= 15

        c.line(50, y, 545, y)
        y -= 20
        c.setFont("Helvetica-Bold", 11)
        c.drawString(400, y, f": ¥{total:,}")

        # --- QRURL---
        qr = QR()
        renderer.draw(qr, f"https://www.example.com/orders/{order_no}",
                      x=50, y=50, width=60, height=60)
        c.setFont("Helvetica", 7)
        c.drawString(50, 40, "")

        c.save()

#
items = [
    ("4901234567894", "A", 3, 1500),
    ("4912345678904", "B", 1, 2800),
    ("4923456789014", "C", 5, 680),
]
create_delivery_note("delivery_note.pdf", "ORD-2026-00123", items)
```

4.3 実例2: 商品ラベル一括印刷

複数の JAN コードラベルを1枚のA4用紙に並べて印刷します。ラベルシートへの印刷に最適。

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from pao_barcode import JAN13
from pao_barcode.renderers import ReportLabRenderer

def create_product_labels(file_path, products, cols=3, rows=8):
    """A4"""
    page_w, page_h = A4
    c = canvas.Canvas(file_path, pagesize=A4)
    renderer = ReportLabRenderer(c, font_size=6, quiet_zone=0.03)

    jan = JAN13()

    #
    margin_x, margin_y = 15 * mm, 15 * mm
    label_w = (page_w - 2 * margin_x) / cols
    label_h = (page_h - 2 * margin_y) / rows

    for i, (jan_code, product_name, price) in enumerate(products):
        #
        page_idx = i // (cols * rows)
        idx_in_page = i % (cols * rows)
        if idx_in_page == 0 and i > 0:
            c.showPage()

        col = idx_in_page % cols
        row = idx_in_page // cols

        #
        lx = margin_x + col * label_w
        ly = page_h - margin_y - (row + 1) * label_h

        #
        c.setFont("Helvetica", 7)
        c.drawString(lx + 5, ly + label_h - 12, product_name)

        #
        c.setFont("Helvetica-Bold", 9)
        c.drawString(lx + 5, ly + label_h - 24, f"¥{price:,}")

        # JAN
        renderer.draw(jan, jan_code,
                      x=lx + 5, y=ly + 3,
                      width=label_w - 10, height=30)
```

```
    c.save()

#
products = [
    ("4901234567894", " 500ml", 98),
    ("4912345678904", " ", 130),
    ("4923456789014", " ", 248),
    ("4934567890124", " 1L", 198),
    ("4945678901234", " 6", 158),
    ("4956789012344", " 400g", 148),
    # ...
]
create_product_labels("product_labels.pdf", products)
```

> ヒント: cols と rows を変えるだけで、ラベルシートのレイアウトを自由に調整できます。市販のラベル用紙（エーワン等）のサイズに合わせてカスタマイズしてみてください。

4.4 実例3: コンビニ払込票

GS1-128 コンビニバーコードを含む払込票です。draw_gs1_128_convenience メソッドを使うと、2行テキスト（金額情報行 + 企業コード行）を自動生成します。

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from pao_barcode import GS1_128
from pao_barcode.renderers import ReportLabRenderer

def create_convenience_slip(file_path, payment_data):
    """PDF"""
    page_w, page_h = A4
    c = canvas.Canvas(file_path, pagesize=A4)
    renderer = ReportLabRenderer(c, font_size=7)

    # ---  ---
    c.setFont("Helvetica-Bold", 14)
    c.drawCentredString(page_w / 2, page_h - 50, "")

    # ---  ---
    y = page_h - 90
    c.setFont("Helvetica", 10)
    c.drawString(50, y, f": {payment_data['payee']}")
    c.drawString(50, y - 18, f": ¥{payment_data['amount']:,}")
    c.drawString(50, y - 36, f": {payment_data['due_date']}")

    # ---  ---
    gs1 = GS1_128()
    barcode_data = payment_data['barcode_data']

    # draw_gs1_128_convenience 2
    renderer.draw_gs1_128_convenience(
        gs1, barcode_data,
        x=50, y=page_h - 220,
        width=page_w - 100, height=80
    )

    # ---  ---
    c.setDash(3, 3)
    c.line(30, page_h - 250, page_w - 30, page_h - 250)
    c.setDash()

    c.save()

#
payment_data = {
    'payee': 'oo',
```

```
    'amount': 12500,
    'due_date': '2026-03-31',
    'barcode_data': '{FNC1}9191234500000000000000452087500401310029500',
}
create_convenience_slip("convenience_slip.pdf", payment_data)
```

> ヒント: コンビニバーコードのデータは {FNC1} + 43桁の数字です。44桁目のチェックディジットは自動計算されます。44桁すべてを渡しても構いません。

4.5 実例4: 配送伝票（QR + 1D複合）

1つの帳票にQRコードとCode128を組み合わせた配送伝票。追跡番号をCode128で、追跡URLをQRコードで表現します。

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from pao_barcode import Code128, QR
from pao_barcode.renderers import ReportLabRenderer

def create_shipping_label(file_path, shipment):
    """PDF"""
    page_w, page_h = A4
    c = canvas.Canvas(file_path, pagesize=A4)
    renderer = ReportLabRenderer(c)

    # ---  ---
    c.rect(40, page_h - 350, page_w - 80, 310)

    # ---  ---
    c.setFont("Helvetica-Bold", 14)
    c.drawString(50, page_h - 60, "")

    # ---  ---
    c.setFont("Helvetica", 10)
    y = page_h - 90
    c.drawString(50, y, f": {shipment['to_name']}")
    c.drawString(50, y - 15, f": {shipment['to_address']}")
    c.drawString(50, y - 30, f"TEL: {shipment['to_tel']}")

    # ---  ---
    y -= 55
    c.drawString(50, y, f": {shipment['from_name']}")
    c.drawString(50, y - 15, f": {shipment['from_address']}")

    # --- Code128---
    bc = Code128()
    renderer.draw(bc, shipment['tracking_no'],
                  x=50, y=page_h - 270,
                  width=280, height=50)

    # --- QRURL---
    qr = QR()
    track_url = f"https://track.example.com/{shipment['tracking_no']}"
    renderer.draw(qr, track_url,
                  x=page_w - 150, y=page_h - 280,
                  width=100, height=100)

    c.setFont("Helvetica", 6)
```

```
    c.drawCentredString(page_w - 100, page_h - 290, "")

    # ---  ---
    c.setFont("Helvetica", 8)
    c.drawString(50, page_h - 330, f": {shipment['ship_date']}")

    c.save()

#
shipment = {
    'tracking_no': '1234-5678-9012',
    'to_name': '',
    'to_address': '1-1-1',
    'to_tel': '03-1234-5678',
    'from_name': '',
    'from_address': '2-2-2',
    'ship_date': '2026-02-05',
}
create_shipping_label("shipping_label.pdf", shipment)
```

4.6 実例5: 在庫管理帳票

DataMatrix と Code128 を組み合わせた在庫管理帳票。管理番号をCode128で、GS1コードをDataMatrixで表現します。

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from pao_barcode import Code128, DataMatrix
from pao_barcode.renderers import ReportLabRenderer

def create_inventory_report(file_path, inventory_items):
    """PDF"""
    page_w, page_h = A4
    c = canvas.Canvas(file_path, pagesize=A4)
    renderer = ReportLabRenderer(c, font_size=6)

    bc = Code128()
    dm = DataMatrix()

    # ---  ---
    c.setFont("Helvetica-Bold", 14)
    c.drawCentredString(page_w / 2, page_h - 40, "")
    c.setFont("Helvetica", 8)
    c.drawString(page_w - 150, page_h - 40, "2026-02-05 ")

    # ---  ---
    y = page_h - 75
    c.setFont("Helvetica-Bold", 8)
    headers = [
        (50, ""),
        (130, ""),
        (270, "GS1"),
        (355, ""),
        (460, ""),
        (510, ""),
    ]
    for hx, label in headers:
        c.drawString(hx, y, label)
    c.line(45, y - 5, 555, y - 5)

    # ---  ---
    c.setFont("Helvetica", 7)
    y -= 8
    row_h = 35

    for item_no, gs1_code, name, qty, location in inventory_items:
        y -= row_h

        #
```

```
            c.drawString(50, y + 18, item_no)

            # Code128
            renderer.draw(bc, item_no,
                          x=125, y=y + 2, width=130, height=28,
                          show_text=False)

            # DataMatrixGS1
            renderer.draw(dm, gs1_code,
                          x=270, y=y + 2, width=28, height=28)

            #
            c.drawString(355, y + 18, name)
            c.drawString(460, y + 18, str(qty))
            c.drawString(510, y + 18, location)

            c.line(45, y - 2, 555, y - 2)

        c.save()

    #
    inventory = [
        ("INV-001", "0104901234567894102 1ABC", " 500ml", 240, "A-1-3"),
        ("INV-002", "0104912345678904102 1DEF", " ", 80, "B-2-1"),
        ("INV-003", "0104923456789014102 1GHI", " ", 45, "B-2-2"),
        ("INV-004", "0104934567890124102 1JKL", " 1L", 120, "C-1-1"),
    ]
    create_inventory_report("inventory_report.pdf", inventory)
```

> ヒント: show_text=False を draw() に渡すと、バーコード下のテキストを一時的に非表示にできます。テーブル内など、スペースが限られている場面で便利。

4.7 座標系とサイズ

ReportLab の座標系:

```
(0, 842) ──────────────────── (595, 842)    ← A4
    │                              │
    │     (50, 700)                │
    │     ┌──────────────┐         │
    │     │      │       │     ← draw(bc, data, 50, 700, 200, 50)
    │     │              │         │         = (50, 700)
    │     └──────────────┘         │        200pt, 50pt
    │     (50, 750)                │
    │                              │
(0, 0) ──────────────────── (595, 0)      ← A4
```

よく使うページサイズ:

サイズ	ポイント	mm換算
A4	595 x 842	210 x 297
A3	842 x 1190	297 x 420
B5	516 x 729	182 x 257
Letter	612 x 792	216 x 279

mm → ポイント変換:

```
from reportlab.lib.units import mm
x = 20 * mm   # 20mm → 56.7
```

4.8 日本語フォント

ReportLab で日本語テキストを表示するには、フォントの登録が必要です。

```
from reportlab.pdfgen import canvas
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont

# : MS
pdfmetrics.registerFont(TTFont('MSGothic', 'msgothic.ttc'))

c = canvas.Canvas("japanese.pdf")
c.setFont("MSGothic", 12)
c.drawString(50, 700, "")

#
from pao_barcode import Code128
from pao_barcode.renderers import ReportLabRenderer

renderer = ReportLabRenderer(c, font_name='MSGothic', font_size=8)
bc = Code128()
renderer.draw(bc, "ABC-123", x=50, y=600, width=200, height=50)

c.save()
```

5.1 PillowRendererの基本

```
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()
bc = Code128()

# PIL Image
img = renderer.render(bc, "Hello-2024", width=400, height=100)
img.show()   #
img.save("barcode.png")  # PIL
```

PillowRendererのオプション:

```
renderer = PillowRenderer(
    fore_color=(0, 0, 0),        #  (R,G,B) 0-255:
    back_color=(255, 255, 255),   #  (R,G,B) or None():
    show_text=True,               # : True
    font_path=None,               # TrueType:
    font_size=14,                 # (px): 14
    quiet_zone=0.05,              # : 0.05 (5%)
)
```

座標系: Pillow は原点が左上、Y軸は下向きです。

5.2 ファイル出力（PNG / JPEG / BMP）

```
from pao_barcode import QR
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()
qr = QR()

# PNG ()
renderer.render_to_file(qr, "https://www.pao.ac/", "qr.png", 300, 300)

# JPEG
renderer.render_to_file(qr, "https://www.pao.ac/", "qr.jpg", 300, 300, format='JPEG')

# BMP
renderer.render_to_file(qr, "https://www.pao.ac/", "qr.bmp", 300, 300, format='BMP')
```

5.3 Base64出力（Webアプリ向け）

```
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()
bc = Code128()

# data:URI  Base64
data_uri = renderer.render_to_base64(bc, "HELLO-123", 400, 100)
# → "..."

# HTML
html = f'<img src="{data_uri}" alt="barcode">'

#
png_bytes = renderer.render_to_bytes(bc, "HELLO-123", 400, 100)
# → b'\x89PNG\r\n\x1a\n...'

# JPEG
jpeg_bytes = renderer.render_to_bytes(bc, "HELLO-123", 400, 100, format='JPEG')
```

> ヒント: Flask や FastAPI で <img> タグに直接埋め込むなら render_to_base64()、Response で画像を返すなら render_to_bytes() が便利です。

5.4 既存画像への描画

draw() メソッドを使うと、既存の PIL Image にバーコードを描画できます。

```
from PIL import Image
from pao_barcode import Code128, QR
from pao_barcode.renderers import PillowRenderer

#
canvas = Image.new('RGB', (800, 600), (255, 255, 255))

renderer = PillowRenderer()

# 1
bc = Code128()
renderer.draw(canvas, bc, "ITEM-001", x=50, y=50, width=300, height=80)
renderer.draw(canvas, bc, "ITEM-002", x=50, y=160, width=300, height=80)

qr = QR()
renderer.draw(canvas, qr, "https://www.pao.ac/", x=500, y=50, width=200, height=200)

canvas.save("multi_barcode.png")
```

5.5 透明背景のバーコード

back_color=None で、RGBA モードの透明背景画像を生成します。

```
from pao_barcode import QR
from pao_barcode.renderers import PillowRenderer

#
renderer = PillowRenderer(back_color=None)
qr = QR()

img = renderer.render(qr, "Hello", 200, 200)
# img.mode == 'RGBA'

# PNG
img.save("qr_transparent.png")

#
from PIL import Image
background = Image.open("background.jpg").convert("RGBA")
background.paste(img, (100, 100), img)  # 3
background.save("composited.png")
```

> ヒント:
透明背景のバーコードは、名刺やポスターのデザインに重ねるときに便利。背景色を気にせず合成できます。

5.6 色のカスタマイズ

```
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

#  +
renderer = PillowRenderer(
    fore_color=(0, 0, 128),
    back_color=(255, 255, 240),
)
bc = Code128()
img = renderer.render(bc, "COLOR-TEST", 400, 100)
img.save("color_barcode.png")

#  +
renderer = PillowRenderer(
    fore_color=(200, 0, 0),
    back_color=None,
)
img = renderer.render(bc, "RED-BARCODE", 400, 100)
img.save("red_barcode.png")
```

6.1 1次元バーコード

Code39

```
from pao_barcode import Code39
from pao_barcode.renderers import PillowRenderer

bc = Code39()
bc.show_start_stop = True  # /*

renderer = PillowRenderer()
renderer.render_to_file(bc, "HELLO-123", "code39.png", 400, 100)
```

入力可能文字: 数字（0-9）、英大文字（A-Z）、記号（- . $ / + % スペース）

Code93

```
from pao_barcode import Code93
from pao_barcode.renderers import PillowRenderer

bc = Code93()

renderer = PillowRenderer()
renderer.render_to_file(bc, "Hello123", "code93.png", 400, 100)
```

入力可能文字: ASCII全文字

Code128

```
from pao_barcode import Code128, CodeSet128
from pao_barcode.renderers import PillowRenderer

bc = Code128()

#
bc.code_mode = CodeSet128.AUTO    #
# bc.code_mode = CodeSet128.CODE_A  #  +  +
# bc.code_mode = CodeSet128.CODE_B  #  +  +  +
# bc.code_mode = CodeSet128.CODE_C  # 2

renderer = PillowRenderer()
renderer.render_to_file(bc, "Hello-2024", "code128.png", 400, 100)
```

コードモード:

モード	説明
AUTO	自動選択（最短幅になるよう動的計画法で最適化）
CODE_A	制御文字 + 数字 + 英大文字
CODE_B	数字 + 英大文字 + 英小文字 + 記号
CODE_C	数字のみ（2桁ずつの高密度エンコード）

制御文字の入力: {CR}, {LF}, {TAB}, {NUL} など

> ヒント: Code128 の AUTO モードは動的計画法で最短幅に最適化します。数字が続く部分は自動的にCODE-C（2桁ペア）に切り替わるので、手動でモード指定する必要はほとんどありません。

NW-7 (Codabar)

```
from pao_barcode import NW7
from pao_barcode.renderers import PillowRenderer

bc = NW7()
bc.show_start_stop = True  # /

renderer = PillowRenderer()
renderer.render_to_file(bc, "A1234567A", "nw7.png", 400, 100)
```

スタート/ストップコード: A, B, C, D

ITF

```
from pao_barcode import ITF
from pao_barcode.renderers import PillowRenderer

bc = ITF()

renderer = PillowRenderer()
renderer.render_to_file(bc, "1234567890", "itf.png", 400, 100)
```

入力: 偶数桁の数字のみ。奇数桁の場合は先頭に0が付加されます。

JAN-13 (EAN-13) / JAN-8 (EAN-8)

```
from pao_barcode import JAN13, JAN8
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()

# JAN-1312 →
jan13 = JAN13()
renderer.render_to_file(jan13, "490123456789", "jan13.png", 300, 100)

# JAN-87 →
jan8 = JAN8()
renderer.render_to_file(jan8, "4901234", "jan8.png", 200, 100)
```

UPC-A / UPC-E

```
from pao_barcode import UPCA, UPCE
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()

# UPC-A11 →
upc_a = UPCA()
renderer.render_to_file(upc_a, "01234567890", "upc_a.png", 300, 100)

# UPC-E7 →
upc_e = UPCE()
renderer.render_to_file(upc_e, "0425261", "upc_e.png", 200, 100)
```

6.2 2次元バーコード

QRコード

```
from pao_barcode import QR
from pao_barcode.renderers import PillowRenderer

qr = QR()
qr.set_error_correction_level("M")  # L/M/Q/H
qr.set_version(0)                    # 0=
# qr.set_encode_mode("AUTO")         # AUTO/NUMERIC/ALPHANUMERIC/BYTE/KANJI

renderer = PillowRenderer()
renderer.render_to_file(qr, "https://www.pao.ac/", "qr.png", 300, 300)
```

誤り訂正レベル:

レベル	復元能力	用途
L	約7%	データ量優先
M	約15%	標準（推奨）
Q	約25%	品質重視
H	約30%	最高品質（ロゴ埋め込み等）

バージョン: 1〜40。0を指定するとデータに応じて自動選択。

DataMatrix

```
from pao_barcode import DataMatrix
from pao_barcode.renderers import PillowRenderer

dm = DataMatrix()
dm.set_code_size("AUTO")         # AUTO / 10x10 / ... / 144x144
dm.set_encode_scheme("AUTO")     # AUTO / ASCII / C40 / TEXT / X12 / EDIFACT / BASE256

renderer = PillowRenderer()
renderer.render_to_file(dm, "Hello World 123", "datamatrix.png", 200, 200)
```

コードサイズ:

- 正方形: 10x10, 12x12, 14x14, 16x16, 18x18, 20x20, 22x22, 24x24, 26x26, 32x32, 36x36, 40x40, 44x44, 48x48, 52x52, 64x64, 72x72, 80x80, 88x88, 96x96, 104x104, 120x120, 132x132, 144x144
- 長方形: 8x18, 8x32, 12x26, 12x36, 16x36, 16x48

PDF417

```
from pao_barcode import PDF417
from pao_barcode.renderers import PillowRenderer

pdf = PDF417()
pdf.set_error_level(2)        # 08 (-1=)
pdf.set_columns(4)            #
pdf.set_rows(0)               #  (0=)
pdf.set_aspect_ratio(3.0)     #
pdf.set_y_height(3)           # Y

renderer = PillowRenderer()
renderer.render_to_file(pdf, "PDF417 Sample Data", "pdf417.png", 400, 150)
```

6.3 GS1系バーコード

GS1-128

```
from pao_barcode import GS1_128
from pao_barcode.renderers import PillowRenderer

gs1 = GS1_128()
renderer = PillowRenderer()

# {FNC1}  {AI}
data = "{FNC1}0100012345678905{AI}10ABC123"
renderer.render_to_file(gs1, data, "gs1_128.png", 500, 120)
```

特殊文字:

記法	意味
{FNC1}	ファンクション1（可変長フィールドの区切り）
{AI}	AI括弧表示（テキスト表示のみ、エンコードには影響しない）

GS1-128 コンビニバーコード

```
from pao_barcode import GS1_128
from pao_barcode.renderers import PillowRenderer

gs1 = GS1_128()
renderer = PillowRenderer()

#
# {FNC1} + 4344
data = "{FNC1}9191234500000000000000452087500401310029500"
renderer.draw_gs1_128_convenience(
    renderer.render(gs1, data, 500, 120),  #
    gs1, data,
    x=0, y=0, width=500, height=120
)

# ReportLab :
from reportlab.pdfgen import canvas
from pao_barcode.renderers import ReportLabRenderer

c = canvas.Canvas("convenience.pdf")
rl_renderer = ReportLabRenderer(c, font_size=7)
rl_renderer.draw_gs1_128_convenience(
```

```
    gs1, data, x=50, y=700, width=400, height=80
)
c.save()
```

GS1 DataBar 標準型

```
from pao_barcode import GS1DataBar14, DataBar14SymbolType
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()

# OMNIDIRECTIONAL
db = GS1DataBar14(symbol_type=DataBar14SymbolType.OMNIDIRECTIONAL)
renderer.render_to_file(db, "1234567890128", "databar14_omni.png", 200, 80)

# STACKED
db = GS1DataBar14(symbol_type=DataBar14SymbolType.STACKED)
renderer.render_to_file(db, "1234567890128", "databar14_stacked.png", 200, 80)

# STACKED_OMNIDIRECTIONAL
db = GS1DataBar14(symbol_type=DataBar14SymbolType.STACKED_OMNIDIRECTIONAL)
renderer.render_to_file(db, "1234567890128", "databar14_stacked_omni.png", 200, 80)
```

GS1 DataBar 限定型

```
from pao_barcode import GS1DataBarLimited
from pao_barcode.renderers import PillowRenderer

db = GS1DataBarLimited()
renderer = PillowRenderer()

#  0  1
renderer.render_to_file(db, "0123456789012", "databar_limited.png", 200, 80)
```

GS1 DataBar 拡張型

```
from pao_barcode import GS1DataBarExpanded, ExpandedSymbolType
from pao_barcode.renderers import PillowRenderer

renderer = PillowRenderer()

# UNSTACKED
db = GS1DataBarExpanded(symbol_type=ExpandedSymbolType.UNSTACKED)
renderer.render_to_file(
    db, "0100012345678905{AI}10ABC123",
```

```
    "databar_expanded.png", 500, 80
)

# STACKED
db = GS1DataBarExpanded(
    symbol_type=ExpandedSymbolType.STACKED,
    num_columns=4
)
renderer.render_to_file(
    db, "0100012345678905{AI}10ABC123",
    "databar_expanded_stacked.png", 300, 120
)
```

6.4 郵便カスタマバーコード

```
from pao_barcode import YubinCustomer
from pao_barcode.renderers import PillowRenderer

yubin = YubinCustomer()
renderer = PillowRenderer(show_text=False)

#  +
renderer.render_to_file(yubin, "264-0025-1-2-503", "yubin.png", 400, 60)
```

入力形式: 郵便番号7桁 + 住所表示番号

注意: 郵便カスタマバーコードにはテキスト表示機能はありません（4-state バーのみ）。

6.5 Webフレームワーク連携

Flask

```
from flask import Flask, Response, jsonify
from pao_barcode import Code128, QR
from pao_barcode.renderers import PillowRenderer

app = Flask(__name__)

@app.route("/barcode/<code>")
def barcode_png(code):
    """Code128PNG"""
    renderer = PillowRenderer()
    bc = Code128()
    png_bytes = renderer.render_to_bytes(bc, code, 400, 100)
    return Response(png_bytes, mimetype="image/png")

@app.route("/qr/<data>")
def qr_png(data):
    """QRPNG"""
    renderer = PillowRenderer()
    qr = QR()
    qr.set_error_correction_level("M")
    png_bytes = renderer.render_to_bytes(qr, data, 300, 300)
    return Response(png_bytes, mimetype="image/png")

@app.route("/barcode-base64/<code>")
def barcode_base64(code):
    """Base64JSON API"""
    renderer = PillowRenderer()
    bc = Code128()
    data_uri = renderer.render_to_base64(bc, code, 400, 100)
    return jsonify({"barcode": data_uri})

if __name__ == "__main__":
    app.run()
```

Django

```
# views.py
from django.http import HttpResponse
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

def barcode_view(request, code):
    renderer = PillowRenderer()
    bc = Code128()
    png_bytes = renderer.render_to_bytes(bc, code, 400, 100)
    return HttpResponse(png_bytes, content_type="image/png")
```

FastAPI

```
from fastapi import FastAPI
from fastapi.responses import Response
from pao_barcode import Code128
from pao_barcode.renderers import PillowRenderer

app = FastAPI()

@app.get("/barcode/{code}")
def barcode(code: str):
    renderer = PillowRenderer()
    bc = Code128()
    png_bytes = renderer.render_to_bytes(bc, code, 400, 100)
    return Response(content=png_bytes, media_type="image/png")
```

> ヒント: Web API として公開する場合、PillowRenderer のインスタンスはリクエストごとに作成するのが安全です。スレッドセーフを気にする必要がありません。

7.1 共通メソッド（全バーコード）

全てのバーコードクラス（BarcodeBase を継承するクラス）で使用できるメソッドです。

set_output_format(format)

出力形式を設定します（SVGモードの場合に使用）。

パラメータ	型	説明
format	str	"png" または "svg"

```
bc.set_output_format("svg")
```

デフォルト: "png"

set_foreground_color(r, g, b, a=255)

前景色（バーの色）を設定します。

パラメータ	型	説明
r	int	赤成分（0-255）
g	int	緑成分（0-255）
b	int	青成分（0-255）
a	int	透明度（0-255、255=不透明）。デフォルト: 255

```
bc.set_foreground_color(0, 0, 128, 255)  #
```

デフォルト: (0, 0, 0, 255)（黒）

set_background_color(r, g, b, a=255)

背景色を設定します。

パラメータ	型	説明
r	int	赤成分（0-255）
g	int	緑成分（0-255）
b	int	青成分（0-255）
a	int	透明度（0-255、255=不透明）。デフォルト: 255

```
bc.set_background_color(255, 255, 240, 255)  #
```

デフォルト: (255, 255, 255, 255)（白）

set_fit_width(fit)

指定幅にフィットさせるかどうかを設定します。

パラメータ	型	説明
fit	bool	True: フィットさせる, False: させない

```
bc.set_fit_width(True)
```

デフォルト: False

set_px_adjust_black(adjust) / set_px_adjust_white(adjust)

黒バー/白バーのピクセル調整値を設定します。印刷時のにじみ補正に使用します。

パラメータ	型	説明
adjust	int	ピクセル調整値

```
bc.set_px_adjust_black(1)   # 1
bc.set_px_adjust_white(-1)  # 1
```

デフォルト: 0

> ヒント: インクジェットプリンターで印刷するとバーがにじんで太くなりがちです。set_px_adjust_black(-1) で微調整すると、スキャナの読み取り精度が上がることがあります。

draw(code, x, y, width, height)

SVGモードでバーコードを描画します。描画結果は get_svg() で取得します。

パラメータ	型	説明
code	str	バーコードデータ
x	int	X座標
y	int	Y座標
width	int	幅（ピクセル）
height	int	高さ（ピクセル）

```
bc.set_output_format("svg")
bc.draw("1234567890", 0, 0, 400, 100)
svg = bc.get_svg()
```

戻り値: bool

get_svg()

SVGモードで描画した結果をSVG文字列として取得します。

```
svg_string = bc.get_svg()
```

戻り値: str （SVG XML文字列）

注意: set_output_format("svg") を設定した後、draw() を呼んでから使用してください。

get_image_base64()

PNGモードで描画した結果をBase64データURI文字列として取得します。

```
bc.draw("1234567890", 0, 0, 400, 100)
data_uri = bc.get_image_base64()
# → "..."
```

戻り値: str （data:URI 形式）

7.2 1次元バーコード共通メソッド

1次元バーコードクラス（BarcodeBase1D を継承するクラス）で使用できるメソッドです。

encode(code)

バーコードデータをエンコードし、バー/スペース幅のパターンを返します。

パラメータ	型	説明
code	str	バーコードデータ

```
pattern = bc.encode("1234567890")
# → [2, 1, 1, 2, 3, 2, ...]  ← bar, space, bar, ...
```

戻り値: List[int] — バーとスペースの幅が交互に並んだリスト（barから開始）

get_pattern(code)

encode() のエイリアスです。同じ結果を返します。

set_show_text(show)

バーコード下の人間可読テキストの表示/非表示を設定します（SVGモード時）。

パラメータ	型	説明
show	bool	True: 表示, False: 非表示

```
bc.set_show_text(True)
```

デフォルト: True

set_text_font_scale(scale)

テキストのフォントサイズ倍率を設定します。

パラメータ	型	説明
scale	float	倍率（0.5〜2.0等）

```
bc.set_text_font_scale(1.2)  # 20%
```

デフォルト: 1.0

set_text_even_spacing(even_spacing)

テキストの均等割り付けを設定します。

パラメータ	型	説明
even_spacing	bool	True: 均等割り付け / False: センタリング

```
bc.set_text_even_spacing(True)   #
bc.set_text_even_spacing(False)  #
```

デフォルト:

- JAN/UPC以外の1Dバーコード: True（均等割り付け）
- JAN-8, JAN-13, UPC-A, UPC-E: False（ガードバー間配置）

GS1-128 コンビニバーコード（2行テキスト）の場合:

- True: 1行目を均等割り付けし、その文字間隔を2行目にも適用
- False: 左寄せで詰めて表示

JAN/UPC での extended_guard との組み合わせ:

extended_guard	text_even_spacing	動作
True	False	標準表示。ロングバー＋セクション間テキスト（デフォル
True	True	ロングバー＋セクション間テキストを均等割り付け
False	False	フラットバー＋テキストをセンタリング
False	True	フラットバー＋テキストを均等割り付け

7.3 Code39

クラス: `Code39`

インポート:

```
from pao_barcode import Code39
```

入力可能文字: 数字（0-9）、英大文字（A-Z）、記号（- . $ / + % スペース）

固有属性:

属性	型	説明	デフォルト
show_start_stop	bool	スタート/ストップ文字（*）の表示	True

使用例

```
bc = Code39()
bc.show_start_stop = False  # *

pattern = bc.encode("HELLO-123")
```

7.4 Code93

クラス: `Code93`

インポート:

```
from pao_barcode import Code93
```

入力可能文字: ASCII全文字（0x00～0x7F）

使用例

```
bc = Code93()
pattern = bc.encode("Hello123")
```

7.5 Code128

クラス: `Code128`

インポート:

```
from pao_barcode import Code128, CodeSet128
```

入力可能文字: ASCII全文字

固有属性:

属性	型	説明	デフォルト
code_mode	CodeSet128	コードセットモード	CodeSet128.AUTO

CodeSet128 列挙値:

値	説明
CodeSet128.AUTO	自動選択（動的計画法で最短幅に最適化）
CodeSet128.CODE_A	制御文字 + 数字 + 英大文字
CodeSet128.CODE_B	数字 + 英大文字 + 英小文字 + 記号
CodeSet128.CODE_C	数字のみ（2桁ずつの高密度エンコード）

制御文字:

記法	意味
{CR}	キャリッジリターン
{LF}	ラインフィード
{TAB}	タブ
{NUL}	NULL

使用例

```
bc = Code128()
bc.code_mode = CodeSet128.AUTO  #

#
pattern = bc.encode("Hello-2024")

# AUTO  CODE-C
pattern = bc.encode("1234567890")
```

7.6 GS1-128

クラス: `GS1_128`

インポート:

```
from pao_barcode import GS1_128
```

encode(code)

通常のGS1-128バーコードをエンコードします。

パラメータ	型	説明
code	str	{FNC1} と {AI} を含むデータ文字列

戻り値: List[int]

encode_convenience(code)

コンビニ収納代行用バーコードをエンコードします。

パラメータ	型	説明
code	str	{FNC1} + 43桁の数字（44桁目のCDは自動計算）、または

戻り値: Tuple[List[int], str] — (パターン, 44桁のデータ文字列)

build_convenience_text_lines(data44)

コンビニバーコードの2行テキストを生成します。

パラメータ	型	説明
data44	str	44桁のデータ文字列（encode_convenience の戻り値）

戻り値: Tuple[str, str] — (1行目, 2行目)

calc_check_digit_mod10w3(digits) ※staticmethod

モジュラス10ウェイト3のチェックディジットを計算します。

パラメータ	型	説明
digits	str	数字文字列

戻り値: int — チェックディジット（0-9）

特殊文字:

記法	意味
{FNC1}	ファンクション1（Code128 の特殊コード 102）
{AI}	AI括弧表示用マーカー（テキスト表示のみ）

使用例

```
gs1 = GS1_128()

# GS1-128
pattern = gs1.encode("{FNC1}0100012345678905{AI}10ABC123")

#
pattern, data44 = gs1.encode_convenience(
    "{FNC1}9191234500000000000000452087500401310029500"
)
line1, line2 = gs1.build_convenience_text_lines(data44)
```

7.7 NW-7（Codabar）

クラス: `NW7`

インポート:

```
from pao_barcode import NW7
```

入力可能文字: 数字（0-9）、記号（- $: / . +）、スタート/ストップ（A, B, C, D）

固有属性:

属性	型	説明	デフォルト
show_start_stop	bool	スタート/ストップコードの表示	True

使用例

```
bc = NW7()
bc.show_start_stop = True
pattern = bc.encode("A1234567A")
```

7.8 ITF

クラス: `ITF`

インポート:

```
from pao_barcode import ITF
```

入力可能文字: 数字（0-9）のみ。偶数桁が必要（奇数桁の場合は先頭に0を付加）。

使用例

```
bc = ITF()
pattern = bc.encode("1234567890")
```

7.9 Matrix 2of5

クラス: `Matrix2of5`

インポート:

```
from pao_barcode import Matrix2of5
```

入力可能文字: 数字（0-9）のみ

使用例

```
bc = Matrix2of5()
pattern = bc.encode("1234567890")
```

7.10 NEC 2of5

クラス: `NEC2of5`

インポート:

```
from pao_barcode import NEC2of5
```

入力可能文字: 数字（0-9）のみ

使用例

```
bc = NEC2of5()
pattern = bc.encode("1234567890")
```

7.11 JAN-8（EAN-8）

クラス: `JAN8`

インポート:

```
from pao_barcode import JAN8
```

入力: 7桁の数字（チェックディジット自動計算）、または8桁（CD含む）

固有属性:

属性	型	説明	デフォルト
extended_guard	bool	ガードバー延長（ロングバー＋セク	True
text_even_spacing	bool	テキスト均等割り付け	False

使用例

```
bc = JAN8()
# : extended_guard=True, text_even_spacing=False
# → JAN
pattern = bc.encode("4901234")    # 7 → CD
pattern = bc.encode("49012348")   # 8CD

#
bc.extended_guard = False

#
bc.text_even_spacing = True
```

7.12 JAN-13（EAN-13）

クラス: `JAN13`

インポート:

```
from pao_barcode import JAN13
```

入力: 12桁の数字（チェックディジット自動計算）、または13桁（CD含む）

固有属性:

属性	型	説明	デフォルト
extended_guard	bool	ガードバー延長（ロングバー＋セク	True
text_even_spacing	bool	テキスト均等割り付け	False

使用例

```
bc = JAN13()
# : extended_guard=True, text_even_spacing=False
# → 1 + 6 + 6
pattern = bc.encode("490123456789")    # 12 → CD
pattern = bc.encode("4901234567894")   # 13CD
```

7.13 UPC-A

クラス: `UPCA`

インポート:

```
from pao_barcode import UPCA
```

入力: 11桁の数字（チェックディジット自動計算）、または12桁（CD含む）

固有属性:

属性	型	説明	デフォルト
extended_guard	bool	ガードバー延長（ロングバー＋セク	True
text_even_spacing	bool	テキスト均等割り付け	False

使用例

```
bc = UPCA()
# : extended_guard=True, text_even_spacing=False
# → 1 + 5 + 5 + 1
pattern = bc.encode("01234567890")   # 11 → CD
```

7.14 UPC-E

クラス: `UPCE`

インポート:

```
from pao_barcode import UPCE
```

入力: 7桁または8桁の数字。UPC-Aからのゼロ圧縮形式。

固有属性:

属性	型	説明	デフォルト
extended_guard	bool	ガードバー延長（ロングバー＋セク	True
text_even_spacing	bool	テキスト均等割り付け	False

使用例

```
bc = UPCE()
# : extended_guard=True, text_even_spacing=False
pattern = bc.encode("0425261")
```

7.15 GS1 DataBar 標準型

クラス: `GS1DataBar14`

インポート:

```
from pao_barcode import GS1DataBar14, DataBar14SymbolType
```

入力: 13桁のGTIN（チェックディジット含む）または12桁（CD自動計算）

コンストラクタ引数:

パラメータ	型	説明	デフォルト
symbol_type	DataBar14SymbolType	シンボルタイプ	OMNIDIRECTIONAL

DataBar14SymbolType 列挙値:

値	説明
OMNIDIRECTIONAL	全方向型（1行）
STACKED	二層型
STACKED_OMNIDIRECTIONAL	全方向二層型

encode(code)

バーコードをエンコードします。STACKED系の場合、patterns および row_heights プロパティに行ごとのパターン情報が格納されます。

戻り値: List[int]

get_human_readable()

人間可読テキスト（GTIN-14形式）を取得します。

戻り値: str

使用例

```
db = GS1DataBar14(symbol_type=DataBar14SymbolType.OMNIDIRECTIONAL)
elements = db.encode("1234567890128")
text = db.get_human_readable()  # "(01)01234567890128"
```

7.16 GS1 DataBar 限定型

クラス: `GS1DataBarLimited`

インポート:

```
from pao_barcode import GS1DataBarLimited
```

入力: 先頭ディジットが 0 または 1 のみ。13桁のGTIN。

encode(code)

戻り値: List[int]

get_human_readable()

戻り値: str

使用例

```
db = GS1DataBarLimited()
elements = db.encode("0123456789012")
text = db.get_human_readable()  # "(01)00123456789012"
```

7.17 GS1 DataBar 拡張型

クラス: `GS1DataBarExpanded`

インポート:

```
from pao_barcode import GS1DataBarExpanded, ExpandedSymbolType
```

入力: AI付きのデータ文字列

コンストラクタ引数:

パラメータ	型	説明	デフォルト
symbol_type	ExpandedSymbolType	シンボルタイプ	UNSTACKED
num_columns	int	列数（STACKED時）	2

ExpandedSymbolType 列挙値:

値	説明
UNSTACKED	一層型
STACKED	多層型

encode(code)

バーコードをエンコードします。STACKED の場合、patterns および row_heights プロパティに行ごとのパターン情報が格納されます。

戻り値: List[int]

get_human_readable()

人間可読テキストを取得します。

戻り値: str

使用例

```
db = GS1DataBarExpanded(
    symbol_type=ExpandedSymbolType.STACKED,
    num_columns=4
)
elements = db.encode("0100012345678905{AI}10ABC123")
text = db.get_human_readable()
```

7.18 郵便カスタマバーコード

クラス: `YubinCustomer`

インポート:

```
from pao_barcode import YubinCustomer
```

入力形式: 郵便番号7桁（ハイフン可） + 住所表示番号（ハイフン区切り）

encode(code)

4-state バーの配列を返します。

パラメータ	型	説明
code	str	住所データ

戻り値: List[int] — バータイプの配列（67本固定）

バータイプ	意味
1	Long（全高）
2	Semi-long upper（上2/3）
3	Semi-long lower（下2/3）
4	Timing bar（中央1/3）

使用例

```
yubin = YubinCustomer()
bars = yubin.encode("264-0025-1-2-503")
# bars = [2, 1, 4, 3, ...]  ← 674-state
```

注意: 郵便カスタマバーコードでは set_show_text, set_text_font_scale はサポートされていません。

7.19 QRコード

クラス: `QR`

インポート:

```
from pao_barcode import QR
```

set_error_correction_level(level)

誤り訂正レベルを設定します。

パラメータ	型	説明
level	str or QRErrorCorrectionLevel	"L", "M", "Q", "H"

レベル	復元能力	用途
L	約7%	データ量優先
M	約15%	標準（推奨）
Q	約25%	品質重視
H	約30%	最高品質

デフォルト: "M"

set_version(version)

QRコードのバージョン（サイズ）を設定します。

パラメータ	型	説明
version	int	0〜40（0=自動）

デフォルト: 0（自動）

set_encode_mode(mode)

エンコードモードを設定します。

パラメータ	型	説明
mode	str or QREncodeMode	"AUTO", "NUMERIC", "ALPHANUMERIC", "BYTE", "KAN

デフォルト: "AUTO"

get_pattern(code)

QRコードの2Dパターンを取得します。

パラメータ	型	説明
code	str or bytes	データ

戻り値: List[List[bool]] — 2Dブーリアン行列。True = 黒モジュール。

get_pattern_string(code)

QRコードパターンを文字列形式で取得します（デバッグ用）。

戻り値: str — "1" = 黒、"0" = 白の文字列

使用例

```
qr = QR()
qr.set_error_correction_level("M")
qr.set_version(0)  #

#
matrix = qr.get_pattern("https://www.pao.ac/")
print(f": {len(matrix)}x{len(matrix[0])}")

#
matrix = qr.get_pattern("".encode("utf-8"))
```

7.20 DataMatrix

クラス: `DataMatrix`

インポート:

```
from pao_barcode import DataMatrix
```

set_code_size(size)

シンボルサイズを設定します。

パラメータ	型	説明
size	str or DxCodeSize	"AUTO", "10x10" ～ "144x144", "8x18" 等

デフォルト: "AUTO"

正方形: 10x10, 12x12, 14x14, 16x16, 18x18, 20x20, 22x22, 24x24, 26x26, 32x32, 36x36, 40x40, 44x44, 48x48, 52x52, 64x64, 72x72, 80x80, 88x88, 96x96, 104x104, 120x120, 132x132, 144x144

長方形: 8x18, 8x32, 12x26, 12x36, 16x36, 16x48

set_encode_scheme(scheme)

エンコードスキームを設定します。

パラメータ	型	説明
scheme	str or DxScheme	"AUTO", "ASCII", "C40", "TEXT", "X12", "EDIFACT", "BAS

デフォルト: "AUTO"

get_pattern(code)

DataMatrix の2Dパターンを取得します。

パラメータ	型	説明
code	str or bytes	データ

戻り値: List[List[bool]]

使用例

```
dm = DataMatrix()
dm.set_code_size("AUTO")
dm.set_encode_scheme("AUTO")
matrix = dm.get_pattern("Hello World 123")
```

7.21 PDF417

クラス: `PDF417`

インポート:

```
from pao_barcode import PDF417
```

set_error_level(level)

誤り訂正レベルを設定します。

パラメータ	型	説明
level	int or PDF417ErrorLevel	-1〜8（-1=自動）

レベル	訂正能力
0	最小
1	低
2	標準（推奨）
3〜8	高〜最大

デフォルト: -1（自動）

set_columns(columns)

列数を設定します。

パラメータ	型	説明
columns	int	1〜30（0=自動）

デフォルト: 0（自動）

set_rows(rows)

行数を設定します。

パラメータ	型	説明
rows	int	3〜90（0=自動）

デフォルト: 0（自動）

set_aspect_ratio(ratio)

縦横比を設定します。

パラメータ	型	説明
ratio	float	縦横比（1.0〜10.0）

デフォルト: 3.0

set_y_height(height)

Y方向（縦）の高さ係数を設定します。

パラメータ	型	説明
height	int	高さ係数（1〜10）

デフォルト: 3

get_pattern(data)

PDF417の2Dパターンを取得します。

パラメータ	型	説明
data	str or bytes	データ

戻り値: List[List[bool]]

使用例

```
pdf = PDF417()
pdf.set_error_level(2)
pdf.set_columns(4)
pdf.set_rows(0)
pdf.set_aspect_ratio(3.0)
pdf.set_y_height(3)

matrix = pdf.get_pattern("PDF417 Sample Data")
print(f": {len(matrix)} x {len(matrix[0])}")
```

7.22 ReportLabRenderer

クラス: `ReportLabRenderer`

PDF帳票のキャンバスにバーコードを直接描画するレンダラーです。

インポート:

```
from pao_barcode.renderers import ReportLabRenderer
```

必要パッケージ: reportlab

コンストラクタ

```
ReportLabRenderer(canvas, **kwargs)
```

パラメータ	型	説明	デフォルト
canvas	reportlab.pdfgen.canvas.Canvas	ReportLabのキャンバス	(必須)
fore_color	Tuple[int,int,int]	前景色 (R,G,B) 0-255	(0, 0, 0)
back_color	Tuple[int,int,int] or None	背景色。Noneで透明	(255, 255, 255)
show_text	bool	テキスト表示	True
font_name	str	フォント名	'Helvetica'
font_size	float	フォントサイズ(pt)	8
quiet_zone	float	クワイエットゾーン比率	0.05

draw(barcode, data, x, y, width, height, **kwargs)

バーコードを描画します。全19タイプに対応。

パラメータ	型	説明
barcode	BarcodeBase	pao_barcodeのエンコーダーインスタンス
data	str	バーコードデータ
x	float	左端X座標（ポイント）
y	float	下端Y座標（ポイント）
width	float	幅（ポイント）
height	float	高さ（ポイント）

kwargs オプション:

キー	型	説明
show_text	bool	コンストラクタ設定の一時上書き
font_size	float	コンストラクタ設定の一時上書き

draw_gs1_128_convenience(barcode, data, x, y, width, height, **kwargs)

GS1-128コンビニバーコードを2行テキスト付きで描画します。

パラメータ	型	説明
barcode	GS1_128	GS1_128インスタンス
data	str	{FNC1} + 43 or 44桁の数字
x, y, width, height	float	描画位置・サイズ

render_to_pdf(barcode, data, file_path, width=200, height=60, **kwargs) ※staticmethod

バーコードを単体PDFファイルとして出力します。1行で完結する便利メソッド。

パラメータ	型	説明
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
file_path	str	出力ファイルパス
width	float	幅（ポイント）。デフォルト: 200
height	float	高さ（ポイント）。デフォルト: 60
**kwargs	ReportLabRendererコンストラクタに渡すオプシ	

```
# PDF1
from pao_barcode import Code128
from pao_barcode.renderers import ReportLabRenderer

ReportLabRenderer.render_to_pdf(Code128(), "1234567890", "barcode.pdf")
```

> ヒント: とりあえずバーコードをPDFにしたいだけなら、この render_to_pdf() 一行で十分。帳票レイアウトは後から考えましょう。

7.23 PillowRenderer

クラス: `PillowRenderer`

PIL Image としてバーコードを描画するレンダラーです。

インポート:

```
from pao_barcode.renderers import PillowRenderer
```

必要パッケージ: Pillow

コンストラクタ

```
PillowRenderer(**kwargs)
```

パラメータ	型	説明	デフォルト
fore_color	Tuple[int,int,int]	前景色 (R,G,B) 0-255	(0, 0, 0)
back_color	Tuple[int,int,int] or None	背景色。Noneで透明(RGBA)	(255, 255, 255)
show_text	bool	テキスト表示	True
font_path	str or None	TrueTypeフォントパス	None (自動検索)
font_size	int	フォントサイズ(px)	14
quiet_zone	float	クワイエットゾーン比率	0.05

render(barcode, data, width, height, **kwargs)

バーコードを新しい PIL Image として生成します。

パラメータ	型	説明
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
width	int	画像幅（ピクセル）
height	int	画像高さ（ピクセル）

戻り値: PIL.Image.Image

render_to_file(barcode, data, file_path, width, height, format=None, **kwargs)

バーコードを画像ファイルとして保存します。

パラメータ	型	説明
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
file_path	str	出力ファイルパス
width	int	画像幅（ピクセル）
height	int	画像高さ（ピクセル）
format	str or None	出力形式（'PNG', 'JPEG', 'BMP'等）。Noneで拡張子から

render_to_bytes(barcode, data, width, height, format='PNG', **kwargs)

バーコードをバイト列として取得します。

パラメータ	型	説明
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
width	int	画像幅（ピクセル）
height	int	画像高さ（ピクセル）
format	str	出力形式。デフォルト: 'PNG'

戻り値: bytes

render_to_base64(barcode, data, width, height, format='PNG', **kwargs)

バーコードを Base64 データURI文字列として取得します。

パラメータ	型	説明
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
width	int	画像幅（ピクセル）
height	int	画像高さ（ピクセル）
format	str	出力形式。デフォルト: 'PNG'

戻り値: str — "..." 形式

draw(image, barcode, data, x, y, width, height, **kwargs)

既存の PIL Image にバーコードを描画します。

パラメータ	型	説明
image	PIL.Image.Image	描画先の画像
barcode	BarcodeBase	エンコーダーインスタンス
data	str	バーコードデータ
x	int	左端X座標（ピクセル）
y	int	上端Y座標（ピクセル）
width	int	幅（ピクセル）
height	int	高さ（ピクセル）

draw_gs1_128_convenience(image, barcode, data, x, y, width, height, **kwargs)

GS1-128コンビニバーコードを2行テキスト付きで既存画像に描画します。

パラメータ	型	説明
image	PIL.Image.Image	描画先の画像
barcode	GS1_128	GS1_128インスタンス
data	str	{FNC1} + 43 or 44桁の数字
x, y, width, height	int	描画位置・サイズ

必須環境

項目	要件
Python	3.8 以上
OS	Windows / macOS / Linux

オプション依存パッケージ

パッケージ	バージョン	用途
reportlab	3.5 以上	PDF帳票出力（ReportLabRenderer）
Pillow	9.0 以上	画像出力（PillowRenderer）

> ヒント: エンコーダー（encode() / get_pattern()）とSVG出力だけなら、外部依存パッケージは不要です。pao_barcode 単体で動きます。

使用許諾

pao_barcode
の使用について、利用者様と有限会社パオ・アット・オフィス（以下「弊社」）は、以下の各項目に同意するものとします。

1. 使用許諾書

この使用許諾書は、利用者様がお使いのパソコンにおいて pao_barcode を使用する場合に同意しなければならない契約書です。

2. 同意

利用者様が pao_barcode を使用する時点で、本使用許諾書に同意されたものとします。

3. ライセンスの購入

製品版を使用して開発を行う場合、1 台の開発用コンピュータにつき 1 ライセンスの購入が必要です。お客様環境等、開発コンピュータでないマシンでの使用にはライセンスは不要です（ランタイムライセンスフリー）。

4. 著作権

pao_barcode の著作権は、いかなる場合においても弊社に帰属いたします。

5. 免責

pao_barcode
の使用によって、直接的または間接的に生じたいかなる損害に対しても、弊社は補償賠償の責任を負わないものとします。

6. 禁止事項

pao_barcode
およびその複製物を第三者に譲渡・貸与することはできません。開発ツールとしての再販・再配布を禁止します。ただし、モジュールとして組み込みを行い再販・再配布する場合は問題ございません。

7. 保証の範囲

弊社は pao_barcode の仕様を予告なしに変更することがあります。利用者様への情報提供は弊社 Web サイトにて行います。

8. 適用期間

本使用許諾条件は利用者様が pao_barcode を使用した日より有効です。

ライセンス

pao_barcode は有限会社パオ・アット・オフィスの製品です。

- 1開発ライセンス: 22,000円（税込） / 20,000円（税抜）
- 必要ライセンス数: pao_barcode を利用して開発するパソコンの台数分
- ランタイムライセンス: 無償（開発環境にのみライセンスが必要です）

試用版について:

- 生成されるバーコードに「SAMPLE」の透かしが表示されます
- 機能制限はありません — すべてのバーコードタイプ・レンダラーを試用できます

製品版について:

- 透かしなしでバーコードを生成できます
- ライセンスの詳細は弊社Webサイトをご確認ください

お問い合わせ

有限会社 パオ・アット・オフィス

- Webサイト: https://www.pao.ac/
- 製品ページ: https://www.pao.ac/barcode.python/
- メール: info@pao.ac

保守・保証につきましては、保守・保証に関する規定をご覧ください。

関連製品

製品	特徴
Barcode.Python C++ Import WASM版	WebAssembly経由。Node.js環境で高速処理
Barcode.Python C++ Import Native版	C++ pybind11 バインディング。最速の処理速度
Barcode.wasm	ブラウザ用 WebAssembly バーコードライブラリ
Barcode.net	.NET用バーコードライブラリ
Barcode.jar	Java用バーコードライブラリ
Barcode.php	PHP用バーコードライブラリ

pao_barcode ユーザーズマニュアル

バージョン 1.1