

# pao\_barcode

Pure Python バーコード生成ライブラリ

ユーザースマニュアル

バージョン 1.0

---

有限会社 パオ・アット・オフィス

<https://www.pao.ac/>

# 目次

---

1. はじめに
  - 1.1 pao\_barcodeとは
  - 1.2 特長
  - 1.3 対応バーコード一覧
  - 1.4 他のBarcode.py製品との違い
  - 1.5 デモサイト
2. インストール
  - 2.1 基本インストール
  - 2.2 オプション依存パッケージ
  - 2.3 動作確認
3. クイックスタート
  - 3.1 エンコーダーの基本
  - 3.2 Pillowで画像出力
  - 3.3 ReportLabでPDF出力
  - 3.4 SVGベクター出力（エンコーダー内蔵）
4. ReportLab帳票組み込みガイド
  - 4.1 ReportLabRendererの基本
  - 4.2 実例1: 納品書にバーコードを組み込む
  - 4.3 実例2: 商品ラベルの一括印刷
  - 4.4 実例3: コンビニ払込票
  - 4.5 実例4: 配送伝票（QR + 1Dバーコード複合）
  - 4.6 実例5: 在庫管理帳票
  - 4.7 座標系とサイズの指定方法
  - 4.8 日本語フォントの使用
5. Pillow画像出力ガイド
  - 5.1 PillowRendererの基本
  - 5.2 ファイル出力（PNG / JPEG / BMP）
  - 5.3 Base64出力（Webアプリ向け）
  - 5.4 既存画像への描画
  - 5.5 透明背景のバーコード
  - 5.6 色のカスタマイズ
6. サンプルコード集
  - 6.1 1次元バーコード

- 6.2 2次元バーコード
- 6.3 GS1系バーコード
- 6.4 郵便カスタマバーコード
- 6.5 Webフレームワーク連携

## 7. APIリファレンス

- 7.1 共通メソッド (全バーコード)
- 7.2 1次元バーコード共通メソッド
- 7.3 Code39
- 7.4 Code93
- 7.5 Code128
- 7.6 GS1-128
- 7.7 NW-7 (Codabar)
- 7.8 ITF
- 7.9 Matrix 2of5
- 7.10 NEC 2of5
- 7.11 JAN-8 (EAN-8)
- 7.12 JAN-13 (EAN-13)
- 7.13 UPC-A
- 7.14 UPC-E
- 7.15 GS1 DataBar 標準型
- 7.16 GS1 DataBar 限定型
- 7.17 GS1 DataBar 拡張型
- 7.18 郵便カスタマバーコード
- 7.19 QRコード
- 7.20 DataMatrix
- 7.21 PDF417
- 7.22 ReportLabRenderer
- 7.23 PillowRenderer

## 8. 動作環境

## 9. ライセンス・お問い合わせ

- 1. [はじめに](#1-はじめに)
  - ・ 1.1 [pao\_barcodeとは](#11-pao\_barcodeとは)
  - ・ 1.2 [特長](#12-特長)
  - ・ 1.3 [対応バーコード一覧](#13-対応バーコード一覧)
  - ・ 1.4 [他のBarcode.py製品との違い](#14-他のbarcodepy製品との違い)
  - ・ 1.5 [デモサイト](#15-デモサイト)
- 1. [インストール](#2-インストール)
  - ・ 2.1 [基本インストール](#21-基本インストール)

- ・ 2.2 [オプション依存パッケージ](#22-オプション依存パッケージ)
- ・ 2.3 [動作確認](#23-動作確認)
- 1. [クイックスタート](#3-クイックスタート)
  - ・ 3.1 [エンコーダーの基本](#31-エンコーダーの基本)
  - ・ 3.2 [Pillowで画像出力](#32-pillowで画像出力)
  - ・ 3.3 [ReportLabでPDF出力](#33-reportlabでpdf出力)
  - ・ 3.4 [SVGベクター出力 (エンコーダー内蔵)](#34-svgベクター出力エンコーダー内蔵)
- 1. [ReportLab帳票組み込みガイド](#4-reportlab帳票組み込みガイド)
  - ・ 4.1 [ReportLabRendererの基本](#41-reportlabrendererの基本)
  - ・ 4.2 [実例1: 納品書にバーコードを組み込む](#42-実例1-納品書にバーコードを組み込む)
  - ・ 4.3 [実例2: 商品ラベルの一括印刷](#43-実例2-商品ラベルの一括印刷)
  - ・ 4.4 [実例3: コンビニ払込票](#44-実例3-コンビニ払込票)
  - ・ 4.5 [実例4: 配送伝票 (QR + 1Dバーコード複合)](#45-実例4-配送伝票qr--1dバーコード複合)
  - ・ 4.6 [実例5: 在庫管理帳票](#46-実例5-在庫管理帳票)
  - ・ 4.7 [座標系とサイズの指定方法](#47-座標系とサイズの指定方法)
  - ・ 4.8 [日本語フォントの使用](#48-日本語フォントの使用)
- 1. [Pillow画像出力ガイド](#5-pillow画像出力ガイド)
  - ・ 5.1 [PillowRendererの基本](#51-pillowrendererの基本)
  - ・ 5.2 [ファイル出力 (PNG / JPEG / BMP)](#52-ファイル出力png--jpeg--bmp)
  - ・ 5.3 [Base64出力 (Webアプリ向け)](#53-base64出力webアプリ向け)
  - ・ 5.4 [既存画像への描画](#54-既存画像への描画)
  - ・ 5.5 [透明背景のバーコード](#55-透明背景のバーコード)
  - ・ 5.6 [色のカスタマイズ](#56-色のカスタマイズ)
- 1. [サンプルコード集](#6-サンプルコード集)
  - ・ 6.1 [1次元バーコード](#61-1次元バーコード)
  - ・ 6.2 [2次元バーコード](#62-2次元バーコード)
  - ・ 6.3 [GS1系バーコード](#63-gs1系バーコード)
  - ・ 6.4 [郵便カスタマバーコード](#64-郵便カスタマバーコード)
  - ・ 6.5 [Webフレームワーク連携](#65-webフレームワーク連携)
- 1. [APIリファレンス](#7-apiリファレンス)
  - ・ 7.1 [共通メソッド (全バーコード)](#71-共通メソッド全バーコード)
  - ・ 7.2 [1次元バーコード共通メソッド](#72-1次元バーコード共通メソッド)
  - ・ 7.3 [Code39](#73-code39)
  - ・ 7.4 [Code93](#74-code93)
  - ・ 7.5 [Code128](#75-code128)
  - ・ 7.6 [GS1-128](#76-gs1-128)
  - ・ 7.7 [NW-7 (Codabar)](#77-nw-7codabar)
  - ・ 7.8 [ITF](#78-itf)
  - ・ 7.9 [Matrix 2of5](#79-matrix-2of5)
  - ・ 7.10 [NEC 2of5](#710-nec-2of5)
  - ・ 7.11 [JAN-8 (EAN-8)](#711-jan-8ean-8)
  - ・ 7.12 [JAN-13 (EAN-13)](#712-jan-13ean-13)
  - ・ 7.13 [UPC-A](#713-upc-a)

- ・ 7.14 [UPC-E](#714-upc-e)
  - ・ 7.15 [GS1 DataBar 標準型](#715-gs1-databar-標準型)
  - ・ 7.16 [GS1 DataBar 限定型](#716-gs1-databar-限定型)
  - ・ 7.17 [GS1 DataBar 拡張型](#717-gs1-databar-拡張型)
  - ・ 7.18 [郵便カスタマバーコード](#718-郵便カスタマバーコード)
  - ・ 7.19 [QRコード](#719-qrコード)
  - ・ 7.20 [DataMatrix](#720-datamatrix)
  - ・ 7.21 [PDF417](#721-pdf417)
  - ・ 7.22 [ReportLabRenderer](#722-reportlabrenderer)
  - ・ 7.23 [PillowRenderer](#723-pillowrenderer)
1. [動作環境](#8-動作環境)
  1. [ライセンス・お問い合わせ](#9-ライセンスお問い合わせ)

# 1. はじめに

## 1.1 pao\_barcode とは

**pao\_barcode** は、Pure Python で実装さ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なバーコードエンジンを Python に移植したもので、**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系統の出力方式をサポートしています。

Pure Python 実装のため、コンパイル済みバイナリに依存せず、**Windows / macOS / Linux**のいずれの環境でも動作します。

## 1.2 特長

特長	説明
<b>Pure Python</b>	C拡張やWASMに依存しない純粋なPython実装。どの環境でもそのまま動作
<b>PDF帳票に直接描画</b>	ReportLabRenderer により、PDF帳票のキャンパスにバーコードを直接描画可能。納品書・請求書・ラベルなどの業務帳票に最適
<b>画像出力</b>	PillowRenderer により、PNG / JPEG / BMP 画像ファイル出力、Base64エンコード、メモリ上での画像生成に対応
<b>SVGベクター出力</b>	エンコーダー内蔵のSVG出力機能で、拡大しても劣化しないベクター画像を生成
<b>全19種バーコード</b>	1次元・2次元・GS1系・郵便バーコードまで幅広く対応
<b>豊富なカスタマイズ</b>	色、サイズ、テキスト表示、クワイエットゾーンなど細かく調整可能
<b>コンビニ払込票対応</b>	GS1-128の標準料金代理収納用バーコード（コンビニバーコード）に完全対応

## 1.3 対応バーコード一覧

### 1次元バーコード（11種類）

バーコード	クラス名	用途・特徴
Code39	Code39	工業用途で広く使用。英数字対応
Code93	Code93	Code39の高密度版。ASCII全文字対応
Code128	Code128	高密度。ASCII全文字対応。物流で多用
GS1-128	GS1_128	物流・医療。AI（アプリケーション識別子）対応
NW-7 (Codabar)	NW7	宅配便、図書館で使用
ITF	ITF	段ボール箱の物流管理。数字のみ
Matrix 2of5	Matrix2of5	工業用途
NEC 2of5	NEC2of5	日本の工業用途

バーコード	クラス名	用途・特徴
JAN-8 (EAN-8)	JAN8	小型商品用。8桁
JAN-13 (EAN-13)	JAN13	日本の商品コード。13桁
UPC-A	UPCA	北米の商品コード。12桁

\*\*注:\*\* UPC-Eは UPC-A のゼロ圧縮版です（下記参照）。

### UPCバーコード（1種類）

バーコード	クラス名	用途・特徴
UPC-E	UPCE	UPC-Aの短縮版。小型商品用

### GS1 DataBarバーコード（3種類）

バーコード	クラス名	用途・特徴
GS1 DataBar 標準型	GS1DataBar14	生鮮食品。Omni/Stacked対応
GS1 DataBar 限定型	GS1DataBarLimited	小型商品。先頭0または1
GS1 DataBar 拡張型	GS1DataBarExpanded	可変長データ対応

### 2次元バーコード（3種類）

バーコード	クラス名	用途・特徴
QRコード	QR	最も普及。日本語対応。大容量
DataMatrix	DataMatrix	小型部品のマーキング。GS1対応
PDF417	PDF417	運転免許証、航空券。大容量

### 特殊バーコード（1種類）

バーコード	クラス名	用途・特徴
郵便カスタマバーコード	YubinCustomer	日本郵便の住所バーコード

## 1.4 他のBarcode.py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：

製品	パッケージ名	特徴
**Pure Python版** (本製品)	pao_barcode	Pure Python実装。ReportLab/Pillow連携。ソースコード付き
Native版	barcode-pao-native	C++ pybind11バインディング。高速処理
WASM版	barcode-pao-wasm	WebAssembly経由。Node.js必要

### Pure Python版 (pao\_barcode) の利点:

- **\*\*PDF帳票への直接描画\*\***: ReportLabRenderer  
により、既存のPDF帳票生成コードにバーコードを簡単に組み込めます。これはPure Python版のみの機能です。
- **\*\*Pillow連携\*\***: PillowRenderer により、PIL Image  
オブジェクトとして生成・加工でき、画像処理パイプラインに組み込めます。
- **\*\*環境を選ばない\*\***: C/C++コンパイラもNode.jsも不要。pip installだけで完結します。
- **\*\*ソースコード付き\*\***: カスタマイズやデバッグが容易です。
- **\*\*ITF対応\*\***: Pure Python版のみ ITF (Interleaved 2 of 5) バーコードに対応しています。

## 1.5 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/>

全バ  
ーコ  
ード  
タイ  
プの  
生成  
・P  
DF  
帳票  
出力  
・画  
像出  
力を  
お試  
しい  
ただ  
けま  
す。

## 2. インストール

### 2.1 基本インストール

```
pip install pao_barcode
```

#### 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、以下のパッケージを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
```

ReportLab はPDF帳票を生成するためのライブラリです。pao\_barcode の ReportLabRenderer を使用してPDF上にバーコードを描画するために必要です。

#### Pillow (画像出力に必要)

```
pip install Pillow
```

Pillow はPython標準の画像処理ライブラリです。pao\_barcode の PillowRenderer を使用してPNG/JPEG/BMP画像を出力するために必要です。

#### 一括インストール

```
pip install pao_barcode reportlab Pillow
```

### 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. クイックスタート

### 3.1 エンコーダーの基本

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。

```
from pao_barcode import Code128, QR, JAN13

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

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

# JAN: encode() → /
jan = JAN13()
pattern = jan.encode("4901234567894")
```

エンコーダーの出力は数値データのみで、描画は含まれません。描画には `**ReportLabRenderer**` または `**PillowRenderer**` を使用します。

### 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''
```

### 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()
```

### 3.4 SVGベクター出力（エンコーダー内蔵）

SVG出力はエンコーダー単体で行えます（レンダラー不要）。

```
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)
```

## 4. ReportLab帳票組み込みガイド

**pao\_barcode** の最大の特長は、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です。

このセクションでは、実際の業務帳票を想定した実践的なコード例を紹介します。

### 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) です。
- A4用紙は 595 x 842 ポイントです。

```
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%)  
)
```

## 4.2 実例1: 納品書にバーコードを組み込む

注文番号のバーコードを含む納品書PDFを生成する例です。

```

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: 商品ラベルの一括印刷

---

複数の JAN コードラベルを1枚の用紙に並べて印刷する例です。

```

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()
    jan.set_extended_guard(True)

    #
    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)
```

## 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の例です。  
draw\_gs1\_128\_convenience  
メソッドを使用すると、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']}")

    # --- ---
    # GS1-128 {FNC1} + 4344CD
    gs1 = GS1_128()
    barcode_data = payment_data['barcode_data'] # : "{FNC1}919123450000000000000452087500401310029500"

    # 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}919123450000000000000452087500401310029500',
}

create_convenience_slip("convenience_slip.pdf", payment_data)

```

## 4.5 実例4: 配送伝票 (QR + 1Dバーコード複合)

---

1つの帳票にQRコードとCode128を組み合わせた配送伝票の例です。

```
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, 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用帳票の例です。

```
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)

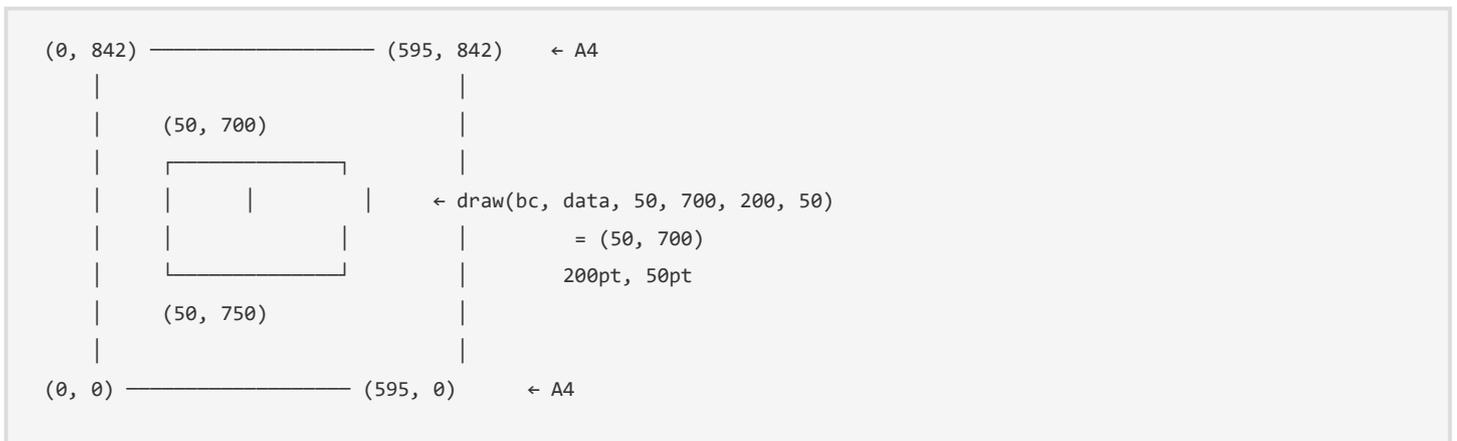
c.save()

#
inventory = [
  ("INV-001", "01049012345678941021ABC", " 500ml", 240, "A-1-3"),
  ("INV-002", "01049123456789041021DEF", " ", 80, "B-2-1"),
  ("INV-003", "01049234567890141021GHI", " ", 45, "B-2-2"),
  ("INV-004", "01049345678901241021JKL", " 1L", 120, "C-1-1"),
]
create_inventory_report("inventory_report.pdf", inventory)

```

## 4.7 座標系とサイズの指定方法

ReportLab の座標系は以下のとおりです：



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 日本語フォントの使用

ReportLab で日本語テキストを表示するには、日本語対応フォントを登録する必要があります。

```
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. Pillow画像出力ガイド

### 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)
# → "data:image/png;base64,iVBORw0KGgo..."

# HTML
html = f''

#
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')
```

## 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 を指定すると、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 PillowRendererer

#
renderer = PillowRendererer(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")
```

## 5.6 色のカスタマイズ

```
from pao_barcode import Code128
from pao_barcode.renderers import PillowRendererer

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

## 6. サンプルコード集

### 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

**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()
jan13.set_extended_guard(True) #
renderer.render_to_file(jan13, "490123456789", "jan13.png", 300, 100)

# JAN-87 →
jan8 = JAN8()
jan8.set_extended_guard(True)
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()
upc_a.set_extended_guard(True)
renderer.render_to_file(upc_a, "01234567890", "upc_a.png", 300, 100)

# UPC-E7 →
upc_e = UPCE()
upc_e.set_extended_guard(True)
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 での使用例

```
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 での使用例

```
# 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 での使用例

```
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")
```

## 7. APIリファレンス

### 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

### 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()
# → "data:image/png;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

## 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 {FNC1} + 44桁の数字

**\*\*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}919123450000000000000000452087500401310029500"
)
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)

**set\_extended\_guard(extended)**

ガードバーの拡張表示を設定します。

パラメータ	型	説明
extended	bool	True: ガードバーを下方に延長

**\*\*デフォルト:\*\*** False

使用例

```
bc = JAN8()
bc.set_extended_guard(True)
pattern = bc.encode("4901234")    # 7 → CD
pattern = bc.encode("49012348")   # 8CD
```

## 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）

### set\_extended\_guard(extended)

JAN-8と同じ。ガードバーの拡張表示を設定します。

**\*\*デフォルト:\*\*** False

### 使用例

```
bc = JAN13()
bc.set_extended_guard(True)
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）

### set\_extended\_guard(extended)

JAN-8と同じ。

### 使用例

```
bc = UPCA()
bc.set_extended_guard(True)
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。

### set\_extended\_guard(extended)

JAN-8と同じ。

### 使用例

```
bc = UPCE()
bc.set_extended_guard(True)
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", "KANJI"

**\*\*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", "BASE256"

**\*\*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。

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ョン	

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

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

## 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で拡張子から推定

### `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 — "data:image/png;base64,..." 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

## 8. 動作環境

### 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)

**\*\*注意:\*\*** エンコーダー (encode() / get\_pattern()) のみを使用する場合は、外部依存パッケージは不要です。

## 9. ライセンス・お問い合わせ

### 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）

- ・ 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。

- ・ 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。

試用  
版に  
つい  
て:

製品  
版に  
つい  
て:

### 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](mailto:info@pao.ac)

保守  
・保  
証に  
つき  
まし  
ては  
、[  
保守  
・保  
証に  
関す  
る規

定](  
http  
s://  
ww  
w.pa  
o.ac  
/pro  
duct  
/sup  
port.  
pdf)  
をご  
覧く  
ださ  
い。

## 関連製品

- ・ **Barcode.py Native版** — C++ pybind11 バインディング版。高速処理が必要な場合に
- ・ **Barcode.py WASM版** — WebAssembly経由。Node.js環境向け
- ・ **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  
\_bar  
cod  
e  
ユー  
ザー  
ズマ  
ニュ  
アル

バージョン 1.0

© 2026 有限会社 パオ・アット・オフィス