Barcode.Python

Python バーコード生成ライブラリ（Native / WASM版）

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

pip install ひとつで、19種のバーコードをPythonから自在に生成。

ユーザーズマニュアル

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 Barcode.Python C++ Import版とは

商品のパッケージ、宅配便の送り状、病院の検体ラベル、工場の部品管理タグ——。

私たちの日常には、驚くほど多くのバーコードが存在しています。

Barcode.Python C++ Import版 は、そのバーコードを Python から簡単に生成 できるライブラリです。

C++で開発された高性能バーコードエンジンを　Python　から直接呼び出す仕組みで、1次元・2次元あわせて　全19種のバーコードを、PNG画像 または SVGベクター で出力できます。

pip install したら、あとはたった3行。

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
result = bc.draw("Hello-2026", 400, 100)
```

これだけで、Base64エンコードされたバーコード画像が手に入ります。

Flask や FastAPI にそのまま組み込めば、あっという間にバーコード生成APIの完成です。

1.2 特長

特長	説明
pip install で即使える	pip install barcode-pao-wasm の1行でセットアップ完了
C++譲りの高速描画	C++で実装されたエンジンが高速にバーコードを生成
全19種バーコード	1D・2D・GS1・郵便まで、業務で必要なバーコードを網羅
PNG / SVG 両対応	画面表示にはPNG、印刷にはSVGと、用途で使い分け可能
ファイル保存もワンステップ	draw_to_file() で PNG/JPEG/GIF/SVG を直接保存
Flask / FastAPI 対応	Webフレームワークにすぐ組み込める実践パターン付き
豊富なカスタマイズ	色、テキスト、バー幅調整、均等割付まで細かく制御可能
コンビニバーコード対応	GS1-128の標準料金代理収納用バーコードにも完全対応

1.3 対応バーコード一覧

1次元バーコード（12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、軍事規格（MIL-STD）にも採用
Code93	Code93	Code39の高密度版。郵便・物流で活用
Code128	Code128	物流の標準。ASCII全文字をエンコード可能
GS1-128	GS1_128	医薬品・物流。ロット番号や有効期限をAIで管理
NW-7 (Codabar)	NW7	宅配便の送り状、図書館の貸出管理でおなじみ
ITF	ITF	段ボール箱のインジケーター。物流の現場で毎日活躍
Matrix 2of5	Matrix2of5	工業用途。数字のみのシンプルな構成
NEC 2of5	NEC2of5	日本の工業現場で使われるバリエーション
JAN-8 (EAN-8)	Jan8	小さな商品用。ガムやキャンディーのパッケージに
JAN-13 (EAN-13)	Jan13	日本の商品バーコードの標準。スーパーのレジで毎日活躍
UPC-A	UPC_A	北米の商品コード。12桁
UPC-E	UPC_E	UPC-Aの短縮版。小さなパッケージに

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	スーパーの青果・精肉売り場。重量や価格を直接エンコー
GS1 DataBar 限定型	GS1DataBarLimited	小型商品向けのコンパクト版
GS1 DataBar 拡張型	GS1DataBarExpanded	可変長データ対応。クーポンや特売情報も格納

郵便バーコード（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	郵便物の住所バーコード。自動区分機で高速仕分け

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	URL、決済、名刺交換——。日本発、世界で最も普及した
DataMatrix	DataMatrix	電子部品の超小型マーキング。GS1ヘルスケアでも標準
PDF417	PDF417	運転免許証、搭乗券。大容量データを1本に集約

1.4 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、用途に応じた2つの製品があります。

C++ Import版（本製品）	Pure Python版（ソースコード付）
パッケージ	barcode-pao-wasm（推奨） barcode-pao-native
こんな方に	手軽にバーコード画像を生成したい
描画方法	draw() で Base64 / SVG を直接取得
出力形式	PNG（Base64）/ SVG
セットアップ	pip install のみ
価格	11,000円（税込）

C++ Import版（本製品）がおすすめな場面:

- Web APIやバッチ処理でバーコード画像を大量生成したい
- Flask / FastAPI でバーコード生成エンドポイントを作りたい
- シンプルに「データを渡して画像を受け取る」だけで十分

Pure Python版がおすすめな場面:

- ReportLab で PDF帳票にバーコードを直接描画したい（納品書、ラベル、帳票など）
- Pillow で画像加工と組み合わせたい
- ソースコード付きで中身を確認・カスタマイズしたい

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。全バーコードの生成をお試しいただけます。

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

2.1 PNG画像出力 — そのまま表示できるBase64

draw() メソッドを呼ぶだけで、Base64エンコードされたPNG画像が返ってきます。

HTMLの <img> タグにそのまま渡すだけ。ファイル保存もサーバー通信も不要です。

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
base64_png = bc.draw("Hello-2026", 400, 100)

# HTML
html = f'<img src="{base64_png}">'
```

PNGが向いている場面:

- Webページでのプレビュー表示
- Flask / FastAPI の JSON レスポンスに含める
- メール本文にインラインで埋め込む

2.2 SVGベクター出力 — 拡大しても美しい

出力形式を "svg" に切り替えるだけで、ベクター形式のSVG文字列が得られます。

どれだけ拡大しても線がぼやけないため、印刷用途に最適です。

```
from barcode_pao_wasm import Code128

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

svg_string = bc.draw("Hello-2026", 400, 100)
# → "<svg xmlns=..."
```

SVGが向いている場面:

- ラベル印刷（拡大しても劣化しない）
- ファイルサイズを小さく抑えたい場合
- CSSやJavaScriptで動的にスタイルを変更したい場合

> ヒント: 同じバーコードオブジェクトで set_output_format() を切り替えれば、PNG版とSVG版の両方を生成できます。プレビューはPNG、ダウンロードはSVG、という使い分けも簡単です。

2.3 ファイル保存 — ワンステップで完了

draw_to_file() メソッドを使えば、PNG / JPEG / GIF / SVG ファイルに直接保存できます。

Base64を経由する必要がないので、バッチ処理に便利です。

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)

# PNG
bc.draw_to_file("Hello-2026", 400, 100, "barcode.png")

# JPEG
bc.set_output_format("jpg")
bc.draw_to_file("Hello-2026", 400, 100, "barcode.jpg")

# SVG
bc.set_output_format("svg")
bc.draw_to_file("Hello-2026", 400, 100, "barcode.svg")
```

2.4 カスタマイズ — 色もテキストも思いのままに

色を変える

前景色（バーの色）と背景色を自由に指定できます。

透明度（アルファ値）にも対応しているので、背景を透明にすることも可能です。

```
#
bc.set_foreground_color(0, 0, 128, 255)
bc.set_background_color(255, 255, 240, 255)

#
bc.set_background_color(0, 0, 0, 0)
```

テキスト表示を調整する

バーコード下部のテキスト（ヒューマンリーダブル）は、表示・非表示だけでなく、サイズや配置まで細かく調整できます。

```
bc.set_show_text(True)              #
bc.set_text_font_scale(1.2)        #
bc.set_text_gap(0.5)               #
bc.set_text_even_spacing(True)     # 1
```

> ヒント: set_text_even_spacing(True) にすると、テキストが各バーの真下に揃って配置されます。見た目がすっきりするので、一般的な1Dバーコードではおすすめです。

バー幅を微調整する（印刷のにじみ対策）

実際に印刷すると、インクのにじみで黒バーが太くなることがあります。

バーコードリーダーの読み取り精度が落ちてしまう場合は、この機能で補正しましょう。

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

幅ぴったり描画

指定した幅にバーコードをぴったり収めたい場合に使います。

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

導入はとてもシンプルです。pip install の1行で完了します。

WASM版 と Native版 の2つのパッケージがありますが、迷ったら WASM版がおすすめ です。

3.1 WASM版のインストール（おすすめ）

```
pip install barcode-pao-wasm
```

必要なもの:

- Python 3.8 以上
- Node.js v14 以上

対応OS: Windows / macOS / Linux

これだけです。C++コンパイラも、ビルドツールも不要。

Node.js さえあれば、どの環境でもすぐに使い始められます。

> ヒント: WASM版は内部で Node.js を使って WebAssembly バイナリを実行しています。Node.jsがインストール済みであれば（最近の開発環境ならほぼ入っていますよね）、追加のセットアップは一切不要です。

3.2 Native版のインストール

```
pip install barcode-pao-native
```

必要なもの:

- Python 3.8 以上
- Visual Studio 2022 以上（C++ Build Tools）

対応OS: Windows のみ

> 注意: Native版はインストール時にC++ソースコードのコンパイルが行われるため、Visual Studio のビルドツールが必要です。環境構築に手間がかかる場合は、WASM版をご検討ください。

3.3 動作確認

インストールが終わったら、さっそく動作を確認してみましょう。

```
# WASM
from barcode_pao_wasm import Code128

# Native
# from barcode_pao_native import Code128

bc = Code128()
bc.set_show_text(True)
result = bc.draw("TEST-1234", 400, 100)
print(f": {result[:50]}...")
# → : ...
```

... で始まる長い文字列が表示されれば成功です。

> ヒント: WASM版とNative版は まったく同じAPIです。import文を変えるだけで、後続のコードは一切変更不要。以降のサンプルではWASM版（barcode_pao_wasm）を使用しますが、Native版でもそのまま動きます。

ここでは、コピー＆ペーストですぐ動くサンプルを紹介します。

4.1 PNG出力（Base64）

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)

base64_png = bc.draw("Hello-2026", 400, 100)
# → "..."

# HTML
html = f'<img src="{base64_png}" alt="">'
```

draw() の戻り値は ... 形式の Data URI 文字列です。

<img> タグの src にそのまま渡せます。

4.2 SVG出力

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_output_format("svg")

svg_string = bc.draw("Hello-2026", 400, 100)
# → "<svg xmlns=..."

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

SVGモードでは <svg xmlns="...">...</svg> 形式の文字列が返ります。

4.3 ファイル保存

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)

# draw_to_file()
bc.draw_to_file("Hello-2026", 400, 100, "barcode.png")

#
bc.set_output_format("svg")
bc.draw_to_file("Hello-2026", 400, 100, "barcode.svg")
```

4.4 バイト列取得（Native版のみ）

Native版では draw_bytes() で PNG のバイナリデータを直接取得できます。

```
from barcode_pao_native import Code128

bc = Code128()
bc.set_show_text(True)

png_bytes = bc.draw_bytes("Hello-2026", 400, 100)
# → b'\x89PNG\r\n\x1a\n...'

with open("barcode.png", "wb") as f:
    f.write(png_bytes)
```

> 注意: draw_bytes() は Native版（barcode-pao-native）のみで利用できます。WASM版には含まれません。

ここからは、バーコードの種類ごとに実践的なサンプルを紹介します。

各バーコードが「どんな場面で使われているか」も添えていますので、用途に合ったバーコードを選ぶ参考にしてください。

5.1 1次元バーコード — 物流・工業の定番

Code39 — 工場で最も古くから使われるバーコード

英数字と一部の記号を表現できます。スタート/ストップコード（*）で囲まれるのが特徴です。

```
from barcode_pao_wasm import Code39

bc = Code39()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_show_start_stop(True)     # *HELLO-123*

result = bc.draw("HELLO-123", 400, 100)
```

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

Code93 — Code39の高密度版

Code39と同じ文字を、より狭いスペースでエンコードできます。さらにASCII全文字に対応。

```
from barcode_pao_wasm import Code93

bc = Code93()
bc.set_show_text(True)
bc.set_text_even_spacing(True)

result = bc.draw("Code93-Test", 400, 100)
```

Code128 — 物流業界の標準

ASCII全文字に対応し、数字は高密度でエンコードできるため、物流伝票で広く使われています。コードモードは通常 "AUTO" にしておけば、最短幅になるよう自動で最適化されます。

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_code_mode("AUTO")    # AUTO / A / B / C

result = bc.draw("Hello-2026", 400, 100)
```

コードモード	説明
"AUTO"	自動で最短幅に最適化（おすすめ）
"A"	制御文字 + 数字 + 英大文字
"B"	数字 + 英大文字 + 英小文字 + 記号
"C"	数字のみ（2桁ずつ高密度エンコード）

> ヒント: 制御文字を入力するには {CR}, {LF}, {TAB} のように中括弧で囲みます。{FNC1} も同様です。

NW-7 (Codabar) — 宅配便の送り状でおなじみ

先頭と末尾にスタート/ストップコード（A/B/C/D）を付けるのがルールです。

```
from barcode_pao_wasm import NW7

bc = NW7()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_show_start_stop(True)

result = bc.draw("A12345B", 400, 100)
```

ITF — 段ボール箱のインジケーター

数字のみ・偶数桁のデータを、バーとスペースに交互にエンコードする方式です。物流の段ボール管理で広く使われています。

```
from barcode_pao_wasm import ITF

bc = ITF()
bc.set_show_text(True)
bc.set_text_even_spacing(True)

result = bc.draw("123456", 400, 100)
```

入力可能: 数字のみ（0-9）。偶数桁 が必要です（奇数桁の場合は先頭に 0 を付加してください）。

Matrix 2of5 / NEC 2of5 — 工業用途の数字専用バーコード

```
from barcode_pao_wasm import Matrix2of5, NEC2of5

# Matrix 2of5
bc = Matrix2of5()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("1234567890", 400, 100)

# NEC 2of5
bc = NEC2of5()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("1234567890", 400, 100)
```

5.2 2次元バーコード — 大容量データを小さな面積に

QRコード — 日本発、世界で最も使われている2Dコード

URL、テキスト、連絡先——なんでも格納できる万能選手です。日本語もそのままエンコードできます。

```
from barcode_pao_wasm import QR

qr = QR()
qr.set_string_encoding("UTF-8")
qr.set_error_correction_level("M")    # L(7%) / M(15%) / Q(25%) / H(30%)
qr.set_version(0)                     # 0=

result = qr.draw("https://www.pao.ac/", 300)
```

誤り訂正レベル	復元能力	こんなときに
"L"	約7%	データ量を最優先したい
"M"	約15%	一般的な用途（おすすめ）
"Q"	約25%	やや過酷な環境（汚れ・傷）
"H"	約30%	ロゴを重ねたい場合にも

DataMatrix — 極小マーキングの世界標準

電子部品やヘルスケア製品の超小型マーキングに使われています。小さくても大容量。

```
from barcode_pao_wasm import DataMatrix

dm = DataMatrix()
dm.set_string_encoding("UTF-8")
dm.set_code_size("AUTO")
dm.set_encode_scheme("AUTO")

result = dm.draw("Hello World", 200)
```

PDF417 — 運転免許証にも使われている大容量コード

1次元バーコードを積み重ねたような構造で、テキスト・数字・バイナリの大量データを格納できます。

```
from barcode_pao_wasm import PDF417

pdf = PDF417()
pdf.set_string_encoding("UTF-8")
pdf.set_error_level(3)
pdf.set_columns(4)
pdf.set_aspect_ratio(0.5)

result = pdf.draw("PDF417 ", 400)
```

5.3 GS1系バーコード — 流通のインフラ

GS1-128 — AI（アプリケーション識別子）で情報を構造化

ロット番号、有効期限、重量——さまざまな情報をAIコードで構造化して格納します。

```
from barcode_pao_wasm import GS1_128

bc = GS1_128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)

result = bc.draw("{FNC1}0100012345678905{AI}10ABC123", 500, 120)
```

特殊文字	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（テキストでAIを括弧で囲んで表示）

コンビニバーコード（標準料金代理収納）

公共料金の払込票に印字されているあのバーコードです。専用メソッド draw_convenience() で生成します。

```
from barcode_pao_wasm import GS1_128

bc = GS1_128()
bc.set_show_text(True)

result = bc.draw_convenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150
)
```

draw_convenience() では2行テキスト（金額情報 + 企業コード）が付いた、標準的なコンビニバーコードが生成されます。44桁目のチェックディジット（モジュラス10ウエイト3）は自動計算されるため、43桁（91で始まる）を入力すればOKです。

GS1 DataBar 標準型 — 青果・精肉売り場で活躍

重量や価格情報をコンパクトにエンコードできるバーコードです。

```
from barcode_pao_wasm import GS1DataBar14

bc = GS1DataBar14()
bc.set_show_text(True)
bc.set_symbol_type("NORMAL")         #
# bc.set_symbol_type("STACKED")         #
# bc.set_symbol_type("STACKED_OMNI")    #

result = bc.draw("4912345678904", 300, 100)
```

GS1 DataBar 限定型 — 小型商品向けコンパクト版

```
from barcode_pao_wasm import GS1DataBarLimited

bc = GS1DataBarLimited()
bc.set_show_text(True)

result = bc.draw("0123456789012", 200, 60)
```

入力: 13桁の数字。先頭桁は 0 または 1 のみ。

GS1 DataBar 拡張型 — クーポンや特売情報も格納可能

可変長データに対応し、多層型（スタック）にも対応しています。

```
from barcode_pao_wasm import GS1DataBarExpanded

bc = GS1DataBarExpanded()
bc.set_show_text(True)

#
bc.set_symbol_type("NORMAL")
result = bc.draw("0100012345678905{AI}10ABC123", 400, 80)

#
bc.set_symbol_type("STACKED")
bc.set_no_of_columns(4)
result = bc.draw("0100012345678905{AI}10ABC123", 300, 150)
```

5.4 商品・郵便バーコード — 身の回りのバーコード

郵便カスタマバーコード — 郵便物を高速仕分け

長さの異なる4種類のバー（ロング・セミアッパー・セミロウワー・タイミング）で住所情報を表現します。

幅はバーの本数から自動計算されるため、高さだけを指定します。

```
from barcode_pao_wasm import YubinCustomer

yubin = YubinCustomer()

#
result = yubin.draw("264-0025-1-2-503", 60)

#
result = yubin.draw_with_width("264-0025-1-2-503", 400, 60)
```

入力形式: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

JAN/EAN バーコード — スーパーのレジで毎日活躍

```
from barcode_pao_wasm import Jan13, Jan8

# JAN-13
jan13 = Jan13()
jan13.set_show_text(True)
jan13.set_extended_guard(True)        #
jan13.set_text_even_spacing(False)    #

result = jan13.draw("490123456789", 300, 100)

# JAN-8
jan8 = Jan8()
jan8.set_show_text(True)
jan8.set_extended_guard(True)
jan8.set_text_even_spacing(False)

result = jan8.draw("4901234", 200, 100)
```

チェックディジットは自動計算されるため、JAN-13なら12桁、JAN-8なら7桁を入力すればOKです。

> ヒント: JAN/UPCバーコードでは set_extended_guard() と set_text_even_spacing() の組み合わせで見た目が変わります。商品バーコードらしい標準的な見た目にするには、extendedGuard=True + textEvenSpacing=False の組み合わせがおすすめです。

UPC バーコード — 北米の商品コード

```
from barcode_pao_wasm import UPC_A, UPC_E

# UPC-A12
upc_a = UPC_A()
upc_a.set_show_text(True)
upc_a.set_extended_guard(True)
upc_a.set_text_even_spacing(False)

result = upc_a.draw("01234567890", 300, 100)

# UPC-E8
upc_e = UPC_E()
upc_e.set_show_text(True)
upc_e.set_extended_guard(True)
upc_e.set_text_even_spacing(False)

result = upc_e.draw("0123456", 200, 100)
```

バーコード生成をWeb APIとして公開するのは、本ライブラリの最も多い使い方のひとつです。

Flask と FastAPI でそのまま使えるパターンを紹介します。

6.1 Flask連携

JSON APIでBase64を返す

```
from flask import Flask, jsonify, request
from barcode_pao_wasm import Code128, QR, Jan13

app = Flask(__name__)

@app.route("/api/barcode", methods=["POST"])
def generate_barcode():
    data = request.json
    barcode_type = data.get("type", "Code128")
    code = data.get("code", "")
    width = data.get("width", 400)
    height = data.get("height", 100)

    classes = {"Code128": Code128, "QR": QR, "Jan13": Jan13}
    bc_class = classes.get(barcode_type, Code128)

    bc = bc_class()
    bc.set_show_text(True)

    if barcode_type == "QR":
        result = bc.draw(code, width)      # 2D size
    else:
        result = bc.draw(code, width, height)

    return jsonify(base64=result)

if __name__ == "__main__":
    app.run(port=5000)
```

HTMLページにバーコードを表示

```
from flask import Flask, render_template_string
from barcode_pao_wasm import Code128

app = Flask(__name__)

@app.route("/barcode/<code>")
def show_barcode(code):
    bc = Code128()
    bc.set_show_text(True)
    base64_png = bc.draw(code, 400, 100)
```

```
    html = f'''
    <html>
    <body>
        <h2>: {code}</h2>
        <img src="{base64_png}" alt="barcode">
    </body>
    </html>
    '''
    return render_template_string(html)

if __name__ == "__main__":
    app.run(port=5000)
```

SVGレスポンス

```
from flask import Flask, Response
from barcode_pao_wasm import Code128

app = Flask(__name__)

@app.route("/barcode/svg/<code>")
def barcode_svg(code):
    bc = Code128()
    bc.set_show_text(True)
    bc.set_output_format("svg")

    svg_string = bc.draw(code, 400, 100)
    return Response(svg_string, mimetype="image/svg+xml")

if __name__ == "__main__":
    app.run(port=5000)
```

PNG バイナリレスポンス（Native版のみ）

```
from flask import Flask, Response
from barcode_pao_native import Code128

app = Flask(__name__)

@app.route("/barcode/png/<code>")
def barcode_png(code):
    bc = Code128()
    bc.set_show_text(True)

    png_bytes = bc.draw_bytes(code, 400, 100)
    return Response(png_bytes, mimetype="image/png")
```

```
if __name__ == "__main__":
    app.run(port=5000)
```

6.2 FastAPI連携

```
from fastapi import FastAPI
from fastapi.responses import JSONResponse, Response
from barcode_pao_wasm import Code128, QR

app = FastAPI()

@app.get("/api/barcode")
def generate_barcode(
    code: str,
    type: str = "Code128",
    width: int = 400,
    height: int = 100,
):
    classes = {"Code128": Code128, "QR": QR}
    bc_class = classes.get(type, Code128)

    bc = bc_class()
    bc.set_show_text(True)

    if type == "QR":
        result = bc.draw(code, width)
    else:
        result = bc.draw(code, width, height)

    return JSONResponse({"base64": result})

@app.get("/api/barcode/svg")
def barcode_svg(code: str, width: int = 400, height: int = 100):
    bc = Code128()
    bc.set_show_text(True)
    bc.set_output_format("svg")

    svg_string = bc.draw(code, width, height)
    return Response(svg_string, media_type="image/svg+xml")
```

> ヒント: Flask でも FastAPI でも、バーコードオブジェクトはリクエストごとに生成するのが安全です。設定が前のリクエストの影響を受けることを防げます。

ここからは、全メソッドの詳細なリファレンスです。

各メソッドのパラメータ、戻り値、デフォルト値を網羅しています。

7.1 共通メソッド（全バーコード）

すべてのバーコード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。

パラメータ	型	説明
format	str	"png", "jpg", "gif", "svg"

```
bc.set_output_format("png")    # PNGBase64—
bc.set_output_format("svg")    # SVG
bc.set_output_format("jpg")    # JPEG
bc.set_output_format("gif")    # GIF
```

デフォルト: "png"

set_foreground_color(r, g, b, a)

前景色（バーの色）を設定します。

パラメータ	型	説明
r	int	赤（0〜255）
g	int	緑（0〜255）
b	int	青（0〜255）
a	int	透明度（0=透明 〜 255=不透明）

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

set_background_color(r, g, b, a)

背景色を設定します。

パラメータ	型	説明
r	int	赤（0〜255）
g	int	緑（0〜255）
b	int	青（0〜255）
a	int	透明度（0=透明 〜 255=不透明）

```
bc.set_background_color(255, 255, 255, 255)    #
bc.set_background_color(255, 255, 240, 255)    #
bc.set_background_color(0, 0, 0, 0)            #
```

描画メソッド

メソッド	戻り値	説明
draw(...)	str	Base64 Data URI またはSVG文字列を返す
draw_to_file(...)	None	ファイルに直接保存
draw_bytes(...)	bytes	PNG/JPG/GIFのバイト列を返す（Native版のみ）

引数の形式はバーコードの種類によって異なります：

カテゴリ	draw	draw_to_file	draw_bytes
1次元	draw(code, width, height)	draw_to_file(code, width, height, file	draw_bytes(code, width, height)
2次元	draw(code, size)	draw_to_file(code, size, filepath)	draw_bytes(code, size)
郵便	draw(code, height)	draw_to_file(code, height, filepath)	—

戻り値（draw）：

- PNG/JPEG/GIF: "..." 形式の Data URI 文字列
- SVG: "<svg xmlns=...>...</svg>" 形式の文字列
- エラー時: 空文字列 ""

7.2 1次元バーコード共通メソッド

1次元バーコード（郵便カスタマバーコードを除く）で共通して使用できるメソッドです。

set_show_text(show)

バーコード下部のテキスト表示を切り替えます。

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

デフォルト: False

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（0.5〜3.0 推奨）

```
bc.set_text_font_scale(1.0)    #
bc.set_text_font_scale(1.5)    # 1.5
bc.set_text_font_scale(0.8)    #
```

デフォルト: 1.0

set_text_gap(scale)

バーコードとテキストの間隔を調整します。

パラメータ	型	説明
scale	float	倍率（0.0〜3.0 推奨）

```
bc.set_text_gap(1.0)    #
bc.set_text_gap(0.5)    #
bc.set_text_gap(2.0)    #
```

デフォルト: 1.0

set_text_even_spacing(even)

テキストの均等割付を設定します。

パラメータ	型	説明
even	bool	True: 均等割付 / False: 中央寄せ

```
bc.set_text_even_spacing(True)     # 1
bc.set_text_even_spacing(False)    #
```

デフォルト:

・JAN/UPC以外: True（均等割付）
・JAN-8, JAN-13, UPC-A, UPC-E: False（セクション間配置）

> 使い分けのコツ: 一般的な1Dバーコード（Code39, Code128など）では True（均等割付）にすると、各文字がバーの真下に揃って読みやすくなります。一方、JAN/UPCバーコードでは False にして set_extended_guard(True) と組み合わせるのが、商品バーコードとしての標準的な見た目です。

set_fit_width(fit)

指定した幅にぴったり収めるかどうかを設定します。

パラメータ	型	説明
fit	bool	True: ぴったり収める / False: 整数ピクセルで描画

デフォルト: False

> 仕組み: True の場合、バーの幅に小数ピクセルを使用して指定幅にぴったり収めます。False の場合は整数ピクセルのみ使用するため、指定幅より若干小さくなることがあります。

set_px_adjust_black(px)

黒バーの幅を微調整します。印刷時のにじみ補正に使います。

パラメータ	型	説明
px	float	調整値（ピクセル）

```
bc.set_px_adjust_black(0)      #
bc.set_px_adjust_black(-1)     # 1px
bc.set_px_adjust_black(1)      # 1px
```

デフォルト: 0

set_px_adjust_white(px)

白スペースの幅を微調整します。

パラメータ	型	説明
px	float	調整値（ピクセル）

```
bc.set_px_adjust_white(0)      #
bc.set_px_adjust_white(1)      # 1px
bc.set_px_adjust_white(-1)     # 1px
```

デフォルト: 0

7.3 Code39

クラス: Code39 — 工業用途の定番バーコード

入力可能文字: 0-9, A-Z, - . $ / + %, スペース

固有メソッド

set_show_start_stop(show)

テキスト表示時にスタート/ストップコード（*）を表示するかどうか。

パラメータ	型	説明
show	bool	True: HELLO / False: HELLO

デフォルト: False

使用例

```
from barcode_pao_wasm import Code39

bc = Code39()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_show_start_stop(True)
result = bc.draw("HELLO-123", 400, 100)
```

7.4 Code93

クラス: Code93 — Code39の高密度版

入力可能文字: ASCII全文字（0x00〜0x7F）

固有メソッド: なし（共通メソッドのみ）

使用例

```
from barcode_pao_wasm import Code93

bc = Code93()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("Code93-test", 400, 100)
```

7.5 Code128

クラス: Code128 — 物流の標準バーコード

入力可能文字: ASCII全文字（0x00〜0x7F）。制御文字は {CR}, {LF}, {TAB}, {FNC1} 等で入力。

固有メソッド

set_code_mode(mode)

パラメータ	型	説明
mode	str	"AUTO", "A", "B", "C"

モード	対応文字
"AUTO"	自動で最短幅に最適化（おすすめ）
"A"	制御文字 + 数字 + 英大文字 + 一部記号
"B"	数字 + 英大文字 + 英小文字 + 記号
"C"	数字のみ（2桁ずつ高密度エンコード）

デフォルト: "AUTO"

> ヒント: "AUTO" モードでは、データの内容を解析して CODE-A / B / C を動的に切り替え、最短幅になるよう自動最適化します。特別な理由がなければ "AUTO" のままで問題ありません。

制御文字一覧

```
{NUL}, {SOH}, {STX}, {ETX}, {EOT}, {ENQ}, {ACK}, {BEL},
{BS}, {HT}, {LF}, {VT}, {FF}, {CR}, {SO}, {SI},
{DLE}, {DC1}, {DC2}, {DC3}, {DC4}, {NAK}, {SYN}, {ETB},
{CAN}, {EM}, {SUB}, {ESC}, {FS}, {GS}, {RS}, {US}, {DEL}
{FNC1}, {FNC2}, {FNC3}, {FNC4}
```

使用例

```
from barcode_pao_wasm import Code128

bc = Code128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_code_mode("AUTO")
result = bc.draw("Hello-2026", 400, 100)
```

7.6 GS1-128

クラス: GS1_128 — GS1標準準拠。物流・医療分野で使用

入力形式: AI（アプリケーション識別子）とデータの組み合わせ

特殊文字	意味
{FNC1}	ファンクション1（可変長AIの区切り）
{AI}	AI括弧表示（テキスト表示時にAIを括弧で囲む）

固有メソッド

draw(code, width, height)

通常のGS1-128バーコードを生成します。

```
result = bc.draw("{FNC1}0100012345678905{AI}10ABC123", 500, 120)
```

draw_convenience(code, width, height)

標準料金代理収納用（コンビニバーコード）を生成します。

```
result = bc.draw_convenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150
)
```

コンビニバーコードのデータ形式:

位置	桁数	内容
1-2	2	AI: 91 固定
3-8	6	企業コード
9-43	35	収納データ（金額・期限等）
44	1	チェックディジット（モジュラス10ウエイト3、自動計算

Code128 から継承：

メソッド	引数	説明
set_code_mode(mode)	"AUTO" / "A" / "B" / "C"	コードセット設定

使用例

```
from barcode_pao_wasm import GS1_128

# GS1-128
bc = GS1_128()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("{FNC1}0100012345678905{AI}10ABC123", 500, 120)

#
bc2 = GS1_128()
bc2.set_show_text(True)
result = bc2.draw_convenience(
    "{FNC1}91912345000000000000004520875004013100295 00", 500, 150
)
```

7.7 NW-7（Codabar）

クラス: NW7 — 宅配便・図書館で使用

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

データ形式: 先頭と末尾にスタート/ストップコード（A/B/C/D）を付ける → A1234567A

固有メソッド

set_show_start_stop(show)

パラメータ	型	説明
show	bool	True: A1234567A / False: 1234567

デフォルト: False

使用例

```
from barcode_pao_wasm import NW7

bc = NW7()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
bc.set_show_start_stop(True)
result = bc.draw("A12345B", 400, 100)
```

7.8 ITF

クラス: ITF — 段ボール箱の物流管理バーコード

入力可能文字: 0-9 のみ

注意: データは 偶数桁 である必要があります。奇数桁の場合は先頭に 0 を付加してください。

固有メソッド: なし

使用例

```
from barcode_pao_wasm import ITF

bc = ITF()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("123456", 400, 100)
```

7.9 Matrix 2of5

クラス: Matrix2of5 — 工業用の数字専用バーコード

入力可能文字: 0-9 のみ

固有メソッド: なし

使用例

```
from barcode_pao_wasm import Matrix2of5

bc = Matrix2of5()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("1234567890", 400, 100)
```

7.10 NEC 2of5

クラス: NEC2of5 — 日本の工業用途向け

入力可能文字: 0-9 のみ

固有メソッド: なし

使用例

```
from barcode_pao_wasm import NEC2of5

bc = NEC2of5()
bc.set_show_text(True)
bc.set_text_even_spacing(True)
result = bc.draw("1234567890", 400, 100)
```

7.11 JAN-8（EAN-8）

クラス: Jan8 — 小型商品用の8桁バーコード

入力: 数字7桁（チェックディジットは自動計算）

固有メソッド

set_extended_guard(extended)

ガードバー（先頭・中央・末尾の区切りバー）を拡張するかどうかを設定します。

True
にすると、ガードバーがテキスト領域まで長く伸び、テキストは左右のセクションに分かれて配置されます。商品バーコードとしての標準的な外観になります。

False にすると、全バーが同じ高さのフラットな外観になります。

パラメータ	型	説明
extended	bool	True: 拡張（標準外観） / False: フラット

デフォルト: True

テキスト表示パターン

JAN/UPC バーコードでは、set_extended_guard() と set_text_even_spacing() の組み合わせで、4種類の見た目を選べます。

extended_guard	text_even_spacing	見た目
True	False	商品バーコードの標準スタイル。 ガードバーが長く伸び、
True	True	ガードバーが長く伸び、テキストは均等割付
False	False	フラットバー + テキスト中央寄せ
False	True	フラットバー + テキスト均等割付

使用例

```
from barcode_pao_wasm import Jan8

bc = Jan8()
bc.set_show_text(True)
bc.set_extended_guard(True)
bc.set_text_even_spacing(False)
result = bc.draw("4901234", 200, 100)
```

7.12 JAN-13（EAN-13）

クラス: Jan13 — 日本の標準的な商品バーコード（13桁）

入力: 数字12桁（チェックディジットは自動計算）

固有メソッド

set_extended_guard(extended)

7.11 JAN-8 と同じです。

デフォルト: True

> JAN-13の特徴:
拡張ガードバー有効時、先頭1桁がバーコード左側にプレフィックスとして表示されます。日本の商品バーコードの「49」や「45」で始まるおなじみの見た目です。

使用例

```
from barcode_pao_wasm import Jan13

bc = Jan13()
bc.set_show_text(True)
bc.set_extended_guard(True)
bc.set_text_even_spacing(False)
result = bc.draw("490123456789", 300, 100)
```

7.13 UPC-A

クラス: UPC_A — 北米の商品コード（12桁）

入力: 数字11桁（チェックディジットは自動計算）

固有メソッド

set_extended_guard(extended)

7.11 JAN-8 と同じです。

デフォルト: True

> UPC-Aの特徴:
拡張ガードバー有効時、先頭1桁（ナンバーシステム）が左側に、末尾1桁（チェックディジット）が右側に、それぞれ小さく表示されます。

使用例

```
from barcode_pao_wasm import UPC_A

bc = UPC_A()
bc.set_show_text(True)
bc.set_extended_guard(True)
bc.set_text_even_spacing(False)
result = bc.draw("01234567890", 300, 100)
```

7.14 UPC-E

クラス: UPC_E — UPC-Aの短縮版（8桁）。小型商品用

入力: 数字7桁（チェックディジットは自動計算）。UPC-Aのゼロ圧縮形式。

固有メソッド

set_extended_guard(extended)

7.11 JAN-8 と同じです。

デフォルト: True

> UPC-Eの特徴:
UPC-Aの「ゼロ圧縮」版です。拡張ガードバー有効時、先頭のナンバーシステム（0）とチェックディジットが左右に表示されます。

使用例

```
from barcode_pao_wasm import UPC_E

bc = UPC_E()
bc.set_show_text(True)
bc.set_extended_guard(True)
bc.set_text_even_spacing(False)
result = bc.draw("0123456", 200, 100)
```

7.15 GS1 DataBar 標準型

クラス: GS1DataBar14 — 生鮮食品向けのコンパクトバーコード

入力: 数字 8〜13桁（チェックディジットは自動計算）

固有メソッド

set_symbol_type(type)

値	説明
"NORMAL"	標準型（どの方向からでも読み取り可能）
"STACKED"	二層型（省スペース）
"STACKED_OMNI"	標準二層型

デフォルト: "NORMAL"

使用例

```
from barcode_pao_wasm import GS1DataBar14

bc = GS1DataBar14()
bc.set_show_text(True)
bc.set_symbol_type("STACKED_OMNI")
result = bc.draw("4912345678904", 300, 100)
```

7.16 GS1 DataBar 限定型

クラス: GS1DataBarLimited — 先頭桁が0または1に限定されたコンパクト版

入力: 数字 13桁（先頭桁は0または1のみ）

固有メソッド: なし

使用例

```
from barcode_pao_wasm import GS1DataBarLimited

bc = GS1DataBarLimited()
bc.set_show_text(True)
result = bc.draw("0123456789012", 200, 60)
```

7.17 GS1 DataBar 拡張型

クラス: GS1DataBarExpanded — 可変長データ対応

入力: AI + データの組み合わせ（{FNC1}, {AI} 使用可能）

固有メソッド

set_symbol_type(type)

値	説明
"NORMAL"	一層型
"STACKED"	多層型（スペースが限られる場合に）

デフォルト: "NORMAL"

set_no_of_columns(columns)

多層型のセグメント数（列数）を設定します。偶数推奨。

デフォルト: 2

使用例

```
from barcode_pao_wasm import GS1DataBarExpanded

bc = GS1DataBarExpanded()
bc.set_show_text(True)

#
bc.set_symbol_type("NORMAL")
result = bc.draw("0100012345678905{AI}10ABC123", 400, 80)

#
bc.set_symbol_type("STACKED")
bc.set_no_of_columns(4)
result = bc.draw("0100012345678905{AI}10ABC123", 300, 150)
```

7.18 郵便カスタマバーコード

クラス: YubinCustomer — 日本郵便の住所バーコード

入力: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

固有メソッド

draw(code, height)

他のバーコードと異なり、幅は自動計算されるため高さのみ指定します。

パラメータ	型	説明
code	str	郵便番号 + 住所表示番号
height	int	画像の高さ（px）

draw_with_width(code, width, height)

幅も指定したい場合はこちらを使います。

パラメータ	型	説明
code	str	郵便番号 + 住所表示番号
width	int	画像の幅（px）
height	int	画像の高さ（px）

> 注意: テキスト関連メソッド（set_show_text(), set_text_font_scale(), set_text_even_spacing() 等）は使用できません。set_foreground_color(), set_background_color(), set_px_adjust_black(), set_px_adjust_white() は使用可能です。

入力例

入力データ	意味
"264-0025-1-2-503"	〒264-0025 1丁目2番地503号
"1050001-13-6"	〒105-0001 13番6号
"2750026-3-29-2-401"	〒275-0026 3丁目29番2号401

使用例

```
from barcode_pao_wasm import YubinCustomer

yubin = YubinCustomer()

#
result = yubin.draw("264-0025-1-2-503", 60)

#
result = yubin.draw_with_width("264-0025-1-2-503", 400, 60)
```

7.19 QRコード

クラス: QR — 日本発、世界で最も普及している2次元バーコード

入力: 数字、英数字、バイナリ、漢字（Shift-JIS）

固有メソッド

draw(code, size)

パラメータ	型	説明
code	str	エンコードするデータ
size	int	画像サイズ（px、正方形）

set_string_encoding(encoding)

値	説明
"UTF-8"	UTF-8（おすすめ）
"Shift_JIS"	Shift-JIS（レガシー環境との互換性が必要な場合）

デフォルト: "UTF-8"

set_error_correction_level(level)

レベル	復元能力	こんなときに
"L"	約7%	データ量優先
"M"	約15%	一般的な用途（おすすめ）
"Q"	約25%	汚れ・傷への耐性が必要
"H"	約30%	最高品質。ロゴ重ね時にも

デフォルト: "M"

set_version(version)

QRコードのバージョン（セルの数 = サイズ）を指定します。

パラメータ	型	説明
version	int	0（自動）～ 40

デフォルト: 0（データに応じた最小バージョンを自動選択）

set_encode_mode(mode)

値	説明
"AUTO"	データ内容に応じて自動選択
"NUMERIC"	数字のみ（最高効率）
"ALPHANUMERIC"	英数字
"BYTE"	バイトデータ
"KANJI"	漢字（Shift-JIS）

デフォルト: "AUTO"

set_fit_width(fit)

指定サイズにフィットさせるかどうか。

デフォルト: False

QRコードバージョンと最大容量（誤り訂正レベルM）

バージョン	モジュール数	数字	英数字	バイト
1	21x21	34	20	14
5	37x37	202	122	84
10	57x57	652	395	271
20	97x97	2061	1249	858
40	177x177	7089	4296	2953

使用例

```
from barcode_pao_wasm import QR

qr = QR()
qr.set_string_encoding("UTF-8")
qr.set_error_correction_level("H")
qr.set_version(0)
qr.set_fit_width(True)
qr.set_output_format("svg")

result = qr.draw("https://www.pao.ac/", 300)
```

7.20 DataMatrix

クラス: DataMatrix — 超小型マーキングの世界標準

入力: ASCII文字、バイナリデータ、GS1データ（{FNC1} で開始）

固有メソッド

draw(code, size)

パラメータ	型	説明
code	str	エンコードするデータ
size	int	画像サイズ（px、正方形）

set_string_encoding(encoding)

"UTF-8"（デフォルト）または "Shift_JIS"

set_code_size(size)

シンボルのセル数を指定します。

主な値	説明
"AUTO"	自動（おすすめ）
"10x10" 〜 "144x144"	正方形
"8x18", "8x32" 等	矩形

デフォルト: "AUTO"

コードサイズ一覧:

正方形: AUTO, 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)

値	説明
"AUTO"	自動選択（おすすめ）
"ASCII"	ASCII
"C40"	英数字
"TEXT"	テキスト（小文字優先）

値	説明
"X12"	ANSI X12 EDI
"EDIFACT"	EDIFACT
"BASE256"	バイナリ

デフォルト: "AUTO"

set_fit_width(fit)

デフォルト: False

GS1-DataMatrix

GS1データを格納する場合は、先頭に {FNC1} を付けます。

```
dm.draw("{FNC1}0100012345678905{FNC1}10ABC123", 200)
```

使用例

```
from barcode_pao_wasm import DataMatrix

dm = DataMatrix()
dm.set_string_encoding("UTF-8")
dm.set_code_size("AUTO")
dm.set_encode_scheme("AUTO")
dm.set_fit_width(True)
dm.set_output_format("svg")

result = dm.draw("Hello World", 200)
```

7.21 PDF417

クラス: PDF417 — 大容量2次元バーコード。運転免許証・搭乗券に使用

入力: テキスト、数字、バイナリ

固有メソッド

draw(code, width)

パラメータ	型	説明
code	str	エンコードするデータ
width	int	画像の幅（px）。高さは自動計算

set_string_encoding(encoding)

"UTF-8"（デフォルト）または "Shift_JIS"

set_error_level(level)

レベル	訂正能力
-1	自動（おすすめ）
0	最小
1	低
2	標準
3〜8	高〜最大

デフォルト: -1（自動）

set_columns(columns)

列数。0=自動、1〜30で指定。 デフォルト: 0

set_rows(rows)

行数。0=自動、3〜90で指定。 デフォルト: 0

set_aspect_ratio(ratio)

縦横比。1.0〜10.0。 デフォルト: 3.0

set_y_height(height)

Y方向の高さ係数。1〜10。 デフォルト: 3

set_fit_width(fit)

デフォルト: False

使用例

```
from barcode_pao_wasm import PDF417

pdf = PDF417()
pdf.set_string_encoding("UTF-8")
pdf.set_error_level(3)
pdf.set_columns(4)
pdf.set_rows(0)
pdf.set_aspect_ratio(3.0)
pdf.set_y_height(3)
pdf.set_fit_width(True)
pdf.set_output_format("svg")

result = pdf.draw("PDF417 ", 400)
```

Barcode.Python C++ Import版には WASM版 と Native版 の2つのパッケージがあります。

APIはまったく同じ。import文を変えるだけで切り替えできます。

```
# WASM
from barcode_pao_wasm import Code128, QR

# NativeAPI
from barcode_pao_native import Code128, QR
```

比較表

項目	WASM版（おすすめ）	Native版
パッケージ名	barcode-pao-wasm	barcode-pao-native
セットアップ	pip install のみ	C++コンパイラが必要
対応OS	Windows / macOS / Linux	Windows のみ
外部依存	Node.js v14+	Visual Studio 2022+
実行速度	十分高速	最高速（C++ 直接実行）
draw_bytes()	なし	利用可能
大量生成	通常の用途には十分	バッチ大量処理に最適

迷ったらWASM版

正直なところ、ほとんどのケースで WASM版がおすすめ です。

WASM版を選ぶ理由:

- pip install と Node.js だけですぐ使える。Visual Studio のインストールは不要
- Windows でも macOS でも Linux でも動く
- 通常の Web アプリケーション用途では速度差を体感することはまずない
- デプロイ先を選ばない。Docker コンテナにも簡単に載せられる

Native版が適しているケース:

- 数万件規模のバーコードをバッチ生成する必要がある
- PNG のバイト列を直接扱いたい（draw_bytes()）
- Windows 専用の環境で、Visual Studio がすでにインストール済み

> ヒント:
開発はWASM版で始めて、速度が課題になったらNative版に切り替える——という進め方もおすすめです。import文を変えるだけなので、移行コストはほぼゼロです。

WASM版 (barcode-pao-wasm)

項目	要件
Python	3.8 以上
OS	Windows / macOS / Linux
ランタイム	Node.js v14 以上
依存ライブラリ	なし

Native版 (barcode-pao-native)

項目	要件
Python	3.8 以上
OS	Windows
ビルドツール	Visual Studio 2022 以上（C++ Build Tools）
依存ライブラリ	pybind11（自動インストール）

使用許諾

Barcode.Python
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 Barcode.Python 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 Barcode.Python 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

Barcode.Python 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

Barcode.Python
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

Barcode.Python
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 Barcode.Python 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 Barcode.Python を使用した日より有効です。

ライセンス

Barcode.Python C++ Import版は有限会社パオ・アット・オフィスの製品です。

項目	内容
1開発ライセンス	11,000円（税込） / 10,000円（税抜）
必要ライセンス数	Barcode.Python 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）
パッケージ共通	1ライセンスでNative版・WASM版の両方を利用可能

試用版:
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種類・設定を自由にお試しいただけます。

製品版: 透かしなしでバーコードを生成できます。

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	対応環境
Barcode.Python Pure Python版（ソースコード付）	Python（ReportLab / Pillow 連携）
Barcode.wasm	JavaScript / TypeScript
Barcode.flutter	Flutter / Dart
Barcode.net	.NET（C#, VB.NET）
Barcode.jar	Java
Barcode.php	PHP

Barcode.Python C++ Import版 ユーザーズマニュアル

バージョン 1.1 — 2026年2月