qrcode 8.2

Latest version

```
pip install qrcode
```

Released: May 1, 2025

QR Code image generator

Verified details

These details have been verified by PyPI

Maintainers

Project description

Pure python QR Code generator

Generate QR codes.

A standard install uses pypng to generate PNG files and can also render QR codes directly to the console. A standard install is just:

```
pip install qrcode
```

For more image functionality, install qrcode with the `pil` dependency so that pillow is installed and can be used for generating images:

maribedran

SmileyChris

Unverified details

*These details have **not** been verified by PyPI*

Project links

Homepage

Meta

- **License:** BSD License, Other/Proprietary License (BSD)
- **Author:** Lincoln Loop
- qr , denso-wave , IEC18004
- **Requires:** Python <4.0, >=3.9
- **Provides-Extra:** `all` , `pil` , `png`

Classifiers

Development Status

- 5 - Production/Stable

Intended Audience

- Developers

License

- OSI Approved :: BSD License

What is a QR Code?

A Quick Response code is a two-dimensional pictographic code used for its fast readability and comparatively large storage capacity. The code consists of black modules arranged in a square pattern on a white background. The information encoded can be made up of any kind of data (e.g., binary, alphanumeric, or Kanji symbols)

Usage

From the command line, use the installed `qr` script:

```
qr "Some text" > test.png
```

Or in Python, use the `make` shortcut function:

```
import qrcode
img = qrcode.make('Some data here')
type(img)  # qrcode.image.pil.PilImage
img.save("some_file.png")
```

Advanced Usage

For more control, use the `QRCode` class. For example:

```
import qrcode
qr = qrcode.QRCode(
    version=1,
    error_correction=qrcode.constants.ERROR_CORRECT_L,
    box_size=10,
    border=4,
)
qr.add_data('Some data')
qr.make(fit=True)
```

Operating System
- OS Independent

Programming Language
- Python
- Python :: 3
- Python :: 3 :: Only
- Python :: 3.9
- Python :: 3.10
- Python :: 3.11
- Python :: 3.12
- Python :: 3.13

Topic
- Multimedia :: Graphics
- Software Development :: Libraries :: Python Modules

The `version` parameter is an integer from 1 to 40 that controls the size of the QR Code (the smallest, version 1, is a 21x21 matrix). Set to `None` and use the `fit` parameter when making the code to determine this automatically.

`fill_color` and `back_color` can change the background and the painting color of the QR, when using the default image factory. Both parameters accept RGB color tuples.

```
img = qr.make_image(back_color=(255, 195, 235), fill_colo
```

The `error_correction` parameter controls the error correction used for the QR Code. The following four constants are made available on the `qrcode` package:

`ERROR_CORRECT_L`

About 7% or less errors can be corrected.

`ERROR_CORRECT_M` (default)

About 15% or less errors can be corrected.

`ERROR_CORRECT_Q`

About 25% or less errors can be corrected.

`ERROR_CORRECT_H`.

About 30% or less errors can be corrected.

The `box_size` parameter controls how many pixels each “box” of the QR code is.

The `border` parameter controls how many boxes thick the border should be (the default is 4, which is the minimum according to the specs).

You can encode as SVG, or use a new pure Python image processor to encode to PNG images.

The Python examples below use the `make` shortcut. The same `image_factory` keyword argument is a valid option for the `QRCode` class for more advanced usage.

SVG

You can create the entire SVG or an SVG fragment. When building an entire SVG image, you can use the factory that combines as a path (recommended, and default for the script) or a factory that creates a simple set of rectangles.

From your command line:

```
qr --factory=svg-path "Some text" > test.svg
qr --factory=svg "Some text" > test.svg
qr --factory=svg-fragment "Some text" > test.svg
```

Or in Python:

```
import qrcode
import qrcode.image.svg

if method == 'basic':
    # Simple factory, just a set of rects.
    factory = qrcode.image.svg.SvgImage
elif method == 'fragment':
    # Fragment factory (also just a set of rects)
    factory = qrcode.image.svg.SvgFragmentImage
else:
    # Combined path factory, fixes white space that may c
    factory = qrcode.image.svg.SvgPathImage

img = qrcode.make('Some data here', image_factory=factory
```

```
qrcode.image.svg.SvgFillImage
qrcode.image.svg.SvgPathFillImage
```

The `QRCode.make_image()` method forwards additional keyword arguments to the underlying ElementTree XML library. This helps to fine tune the root element of the resulting SVG:

```
import qrcode
qr = qrcode.QRCode(image_factory=qrcode.image.svg.SvgPatl
qr.add_data('Some data')
qr.make(fit=True)

img = qr.make_image(attrib={'class': 'some-css-class'})
```

You can convert the SVG image into strings using the `to_string()` method. Additional keyword arguments are forwarded to ElementTrees `tostring()`:

```
img.to_string(encoding='unicode')
```

Pure Python PNG

If Pillow is not installed, the default image factory will be a pure Python PNG encoder that uses *pypng*.

You can use the factory explicitly from your command line:

```
qr --factory=png "Some text" > test.png
```

Or in Python:

```
img = qrcode.make('Some data here', image_factory=PyPNGI
```

Styled Image

Works only with versions >=7.2 (SVG styled images require 7.4).

To apply styles to the QRCode, use the `StyledPilImage` or one of the standard SVG image factories. These accept an optional `module_drawer` parameter to control the shape of the QR Code.

These QR Codes are not guaranteed to work with all readers, so do some experimentation and set the error correction to high (especially if embedding an image).

Other PIL module drawers:

> *doc/module_drawers.png*

For SVGs, use `SvgSquareDrawer`, `SvgCircleDrawer`, `SvgPathSquareDrawer`, or `SvgPathCircleDrawer`.

These all accept a `size_ratio` argument which allows for "gapped" squares or circles by reducing this less than the default of `Decimal(1)`.

The `StyledPilImage` additionally accepts an optional `color_mask` parameter to change the colors of the QR Code, and an optional `embedded_image_path` to embed an image in the center of the code.

Other color masks:

> *doc/color_masks.png*

Here is a code example to draw a QR code with rounded corners, radial gradient and an embedded image:

```
import qrcode
from qrcode.image.styledpil import StyledPilImage
from qrcode.image.styles.moduledrawers.pil import Rounde
```

```
qr.add_data('Some data')

img_1 = qr.make_image(image_factory=StyledPilImage, modu
img_2 = qr.make_image(image_factory=StyledPilImage, colo
img_3 = qr.make_image(image_factory=StyledPilImage, embe
```

Examples

Get the text content from *print_ascii*:

```
import io
import qrcode
qr = qrcode.QRCode()
qr.add_data("Some text")
f = io.StringIO()
qr.print_ascii(out=f)
f.seek(0)
print(f.read())
```

The *add_data* method will append data to the current QR object. To add new data by replacing previous content in the same object, first use clear method:

```
import qrcode
qr = qrcode.QRCode()
qr.add_data('Some data')
img = qr.make_image()
qr.clear()
qr.add_data('New data')
other_img = qr.make_image()
```

Pipe ascii output to text file in command line:

```
qr --ascii "Some data" > "test.txt"
cat test.txt
```

```
# qr "Some data" > test.png
qr --output=test.png "Some data"
```

Change log

8.2 (01 May 2025)

- Optimize QRColorMask apply_mask method for enhanced performance
- Fix typos on StyledPilImage embeded_* parameters. The old parameters with the typos are still accepted for backward compatibility.

8.1 (02 April 2025)

- Added support for Python 3.13.

8.0 (27 September 2024)

- Added support for Python 3.11 and 3.12.
- Drop support for Python <=3.8.
- Change local development setup to use Poetry.
- Testsuite and code quality checks are done through Github Actions.
- Code quality and formatting utilises ruff.
- Removed `typing_extensions` as a dependency, as it's no longer required with having Python 3.9+ as a requirement. having Python 3.9+ as a requirement.
- Only allow high error correction rate (*qrcode.ERROR_CORRECT_H*) when generating QR codes with embedded images to ensure content is readable

7.4.2 (6 February 2023)

- Allow `pypng` factory to allow for saving to a string (like `qr.save("some_file.png")`) in addition to file-like objects.

to mattiasj-axis!

7.4 (1 February 2023)

- Restructure the factory drawers, allowing different shapes in SVG image factories as well.
- Add a `--factory-drawer` option to the `qr` console script.
- Optimize the output for the `SVGPathImage` factory (more than 30% reduction in file sizes).
- Add a `pypng` image factory as a pure Python PNG solution. If `pillow` is *not* installed, then this becomes the default factory.
- The `pymaging` image factory has been removed, but its factory shortcut and the actual PymagingImage factory class now just link to the PyPNGImage factory.

7.3.1 (1 October 2021)

- Improvements for embedded image.

7.3 (19 August 2021)

- Skip color mask if QR is black and white

7.2 (19 July 2021)

- Add Styled PIL image factory, allowing different color masks and shapes in QR codes
- Small performance inprovement
- Add check for border size parameter

7.1 (1 July 2021)

- Add –ascii parameter to command line interface allowing to output ascii when stdout is piped

- Accept RGB tuples in fill_color and back_color
- Add to_string method to SVG images
- Replace inline styles with SVG attributes to avoid CSP issues
- Add Python3.10 to supported versions

7.0 (29 June 2021)

- Drop Python < 3.6 support.

6.1 (14 January 2019)

- Fix short chunks of data not being optimized to the correct mode.
- Tests fixed for Python 3

6.0 (23 March 2018)

- Fix optimize length being ignored in `QRCode.add_data`.
- Better calculation of the best mask pattern and related optimizations. Big thanks to cryptogun!

5.3 (18 May 2016)

- Fix incomplete block table for QR version 15. Thanks Rodrigo Queiro for the report and Jacob Welsh for the investigation and fix.
- Avoid unnecessary dependency for non MS platforms, thanks to Noah Vesely.
- Make `BaseImage.get_image()` actually work.

5.2 (25 Jan 2016)

- Add `--error-correction` option to qr script.
- Fix script piping to stdout in Python 3 and reading non-UTF-8 characters in Python 3.
- Fix script piping in Windows.
- Add some useful behind-the-curtain methods for tinkerers.

5.2.1

- Small fix to terminal output in Python 3 (and fix tests)

5.2.2

- Revert some terminal changes from 5.2 that broke Python 3's real life tty code generation and introduce a better way from Jacob Welsh.

5.1 (22 Oct 2014)

- Make `qr` script work in Windows. Thanks Ionel Cristian Mărieș
- Fixed print_ascii function in Python 3.
- Out-of-bounds code version numbers are handled more consistently with a ValueError.
- Much better test coverage (now only officially supporting Python 2.6+)

5.0 (17 Jun 2014)

- Speed optimizations.
- Change the output when using the `qr` script to use ASCII rather than just colors, better using the terminal real estate.
- Fix a bug in passing bytecode data directly when in Python 3.
- Substation speed optimizations to best-fit algorithm (thanks Jacob Welsh!).
- Introduce a `print_ascii` method and use it as the default for the `qr` script rather than `print_tty`.

5.0.1

- Update version numbers correctly.

4.0 (4 Sep 2013)

- Support pure-python PNG generation (via pymaging) for Python 2.6+ – thanks Adam Wisniewski!
- SVG image generation now supports alternate sizing (the default box size of 10 == 1mm per rectangle).
- SVG path image generation allows cleaner SVG output by combining all QR rects into a single path. Thank you, Viktor Stískala.
- Added some extra simple SVG factories that fill the background white.

4.0.1

- Fix the pymaging backend not able to save the image to a buffer. Thanks ilj!

4.0.2

- Fix incorrect regex causing a comma to be considered part of the alphanumeric set.
- Switch to using setuptools for setup.py.

4.0.3

- Fix bad QR code generation due to the regex comma fix in version 4.0.2.

4.0.4

- Bad version number for previous hotfix release.

3.1 (12 Aug 2013)

- Important fixes for incorrect matches of the alphanumeric encoding mode. Previously, the pattern would match if a single line was alphanumeric only (even if others wern't). Also, the two characters `{` and `}` had snuck in as valid characters. Thanks to Eran Tromer for the report and fix.

most efficient modes.

3.1.1

- Update change log to contain version 3.1 changes. :P
- Give the `qr` script an `--optimize` argument to control the chunk optimization setting.

3.0 (25 Jun 2013)

- Python 3 support.
- Add QRCode.get_matrix, an easy way to get the matrix array of a QR code including the border. Thanks Hugh Rawlinson.
- Add in a workaround so that Python 2.6 users can use SVG generation (they must install `lxml`).
- Some initial tests! And tox support (`pip install tox`) for testing across Python platforms.

2.7 (5 Mar 2013)

- Fix incorrect termination padding.

2.6 (2 Apr 2013)

- Fix the first four columns incorrectly shifted by one. Thanks to Josep Gómez-Suay for the report and fix.
- Fix strings within 4 bits of the QR version limit being incorrectly terminated. Thanks to zhjie231 for the report.

2.5 (12 Mar 2013)

- The PilImage wrapper is more transparent - you can use any methods or attributes available to the underlying PIL Image instance.
- Fixed the first column of the QR Code coming up empty! Thanks to BecoKo.

2.4 (23 Apr 2012)

- Use a pluggable backend system for generating images, thanks to Branko Čibej! Comes with PIL and SVG backends built in.

2.4.1

- Fix a packaging issue

2.4.2

- Added a `show` method to the PIL image wrapper so the `run_example` function actually works.

2.3 (29 Jan 2012)

- When adding data, auto-select the more efficient encoding methods for numbers and alphanumeric data (KANJI still not supported).

2.3.1

- Encode unicode to utf-8 bytestrings when adding data to a QRCode.

2.2 (18 Jan 2012)

- Fixed tty output to work on both white and black backgrounds.
- Added *border* parameter to allow customizing of the number of boxes used to create the border of the QR code

2.1 (17 Jan 2012)

- Added a `qr` script which can be used to output a qr code to the tty using background colors, or to a file via a pipe.

Help

Installing packages
Uploading packages
User guide
Project name retention
FAQs

About PyPI

PyPI Blog
Infrastructure dashboard
Statistics
Logos & trademarks
Our sponsors

Contributing to PyPI

Bugs and feedback
Contribute on GitHub
Translate PyPI
Sponsor PyPI
Development credits

Using PyPI

Terms of Service
Report security issue
Code of conduct
Privacy Notice
Acceptable Use Policy

Status: All Systems Operational

Developed and maintained by the Python community, for the Python community.
Donate today!

Site map

Switch to desktop version