

# Generate Beautiful QR Codes With Python

by Sarah Haq ⌚ Sep 06, 2023 💬 1 Comment

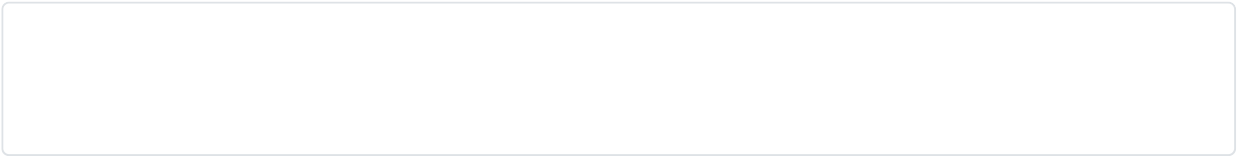
🔑 **intermediate**

Mark as Completed 📌

🐦 Tweet    📱 Share    ✉ Email

## Table of Contents

- [Using Python to Generate a Basic QR Code](#)
- [Changing the Size the QR Code](#)
- [Formatting the Border of the QR Code](#)
- [Changing the Colors of the QR Code](#)
- [Rotating the QR Code](#)
- [Creating Animated QR Codes](#)
- [Conclusion](#)



🔍 [Remove ads](#)

From restaurant e-menus to airline boarding passes, **QR codes** have numerous applications that impact your day-to-day life and enrich the user’s experience. Wouldn’t it be great to make them look good, too? With the help of this tutorial, you’ll learn how to use Python to generate beautiful QR codes for your personal use case.

In its most basic format, a QR code contains black squares and dots on a white background, with information that any smartphone or device with a dedicated QR scanner can decode. Unlike a traditional bar code, which holds information horizontally, a QR code holds the data in two dimensions, and it can hold over a hundred times more information.

### In this tutorial, you’ll learn how to:

- **Generate** a basic black-and-white QR code
- Change the **size** and **margins** of the QR code
- Create **colorful** QR codes

— FREE Email Series —

 **Python Tricks** 

```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```

Email...

Get Python Tricks »

🔒 No spam. Unsubscribe any time.

- 🔍 **Browse Topics**
- 📖 **Guided Learning Paths**
- ⤴ **Basics**    ⤴⤴ **Intermediate**
- ⤴⤴⤴ **Advanced**

- api**   **best-practices**   **career**
- community**   **databases**   **data-science**
- data-structures**   **data-viz**   **devops**
- django**   **docker**   **editors**   **flask**
- front-end**   **gamedev**   **gui**
- machine-learning**   **numpy**   **projects**
- python**   **testing**   **tools**   **web-dev**
- web-scraping**

- **Rotate** the QR code
- Replace the static background with an **animated GIF**

Traditionally, QR codes have been predominantly black-and-white. Now, thanks to creative innovations, QR codes come in all sorts of shapes, sizes, and colors. If you'd like to learn how to create not only two-tone QR codes but also colorful and artistic ones, then this is the tutorial for you.

**Free Bonus:** [Click here to download the sample code](#) that shows you how to generate a beautiful code with Python.

## Using Python to Generate a Basic QR Code

To begin with, you're going to create a black-and-white QR code that encodes some content. To follow along with this tutorial, you'll need to install [Segno](#), which is a popular Python library for generating QR codes. To install Segno, you can run [pip](#) from your [command line](#):

Shell

```
$ python -m pip install segno
```

Once you've installed segno, create a Python file named `basic_qrcode.py`. To create a black-and-white QR code object that encodes some content, you'll have to use the `make_qr()` function. This ensures that you're creating a full-size QR code, and the only mandatory argument you'll need to pass is the information that you want to encode.

**Note:** You can also use the `make()` function to create a QR code object, but depending on the content you're encoding, it might create a micro QR code. For the sake of this tutorial, you can think of that as a smaller QR code. To enforce the creation of a full-size QR code, you should use the `make_qr()` function.

QR codes are capable of handling all types of data, such as alphanumeric characters, symbols, and even URLs. To begin your journey of generating QR codes in Python, you'll start by creating a QR code object that will encode the text "Hello, World":

Python

```
# basic_qrcode.py

import segno

segno.make_qr("Hello, World")
```

At this stage, you've written the code to encode the text "Hello, World" by passing this to the `make_qr()` function. This will greet your user with that text, and it might also offer the option of searching for "Hello, World" on the Internet.

Now, to generate a black-and-white QR code that you can actually view and scan, you'll need to store the encoded content as a variable and use the `.save()` method to save your QR code as an image. Here's an example of how you can create a [variable](#) called `qrcode` that encodes the text "Hello, World" as a black-and-white QR code object, which you then save as a PNG file called `basic_qrcode.png`:

Python

```
# basic_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save("basic_qrcode.png")
```

The `.save()` method serializes the QR code into a file format of your choice, as long as the chosen format is supported. When you apply `.save()` to the variable that you've created with the encoded content, you need to specify the filename including an optional file path. In the example above, you're saving the QR code image as a file named `basic_qrcode.png` in the same directory where you'll be executing your code, so you don't specify a file path.

If you'd like to save the image in a different directory, then you can specify the desired [file path](#) together with the filename as an argument in the `.save()` method.

Once you've finished writing the steps for generating a black-and-white QR code in `basic_qrcode.py`, you'll need to [run the Python script](#) from your command line:

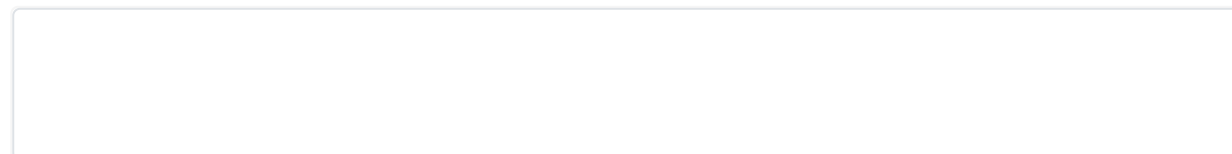
Shell

```
$ python basic_qrcode.py
```

Congratulations, you've just created a black-and-white QR code that encodes the text "Hello, World" using `make_qr()` and `.save()`. You can now scan your QR code image, which you'll find in the same directory that you're running your code from. Or, you can scan the QR code image below:



You may have found that the QR code image is a little difficult to view or read because of its size, so the next item that you're going to adjust is the size of your basic QR code.



 [Remove ads](#)

## Changing the Size the QR Code

When trying to scan your QR code image or the QR code in the tutorial, you might have found it difficult to read because of its size.

If you'd like to to adjust the size of the QR code, then you can add a `scale` parameter to the `.save()` method. The `scale` parameter is a scaling factor that changes the width and height of the QR code image.

For your reference, the default value for `scale` is 1. When you created your first QR code in `basic_qrcode.py`, the size of each black or white square in your QR code was one pixel wide and one pixel high. Each of these black or white squares is called a **module**.

With the code below, you can create a QR code that encodes the content "Hello, World", where each module is 5x5 pixels in size:

Python

```
# scaled_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "scaled_qrcode.png",
    scale=5,
)
```

Here, you've created a Python file named `scaled_qrcode.py` and set `scale=5` within the `.save()` method. To generate this black-and-white QR code with a scaling factor of five, you'll then have to run `scaled_qrcode.py` from the command line:

Shell

```
$ python scaled_qrcode.py
```

Great job! You adjusted the size of your modules by applying the `.save()` method to your QR code object with a `scale` argument of 5. Your QR code should look like this:



It might be a little difficult to see in the image above, but if you enlarge it, then you'll find that the your QR code image also includes some blank space around the QR code. In the next section, you'll learn how to adjust the size or remove this blank space.

## Formatting the Border of the QR Code

To increase the scannability of the QR code and make sure that devices such as smartphones can clearly access the information, Segno puts some blank space around the QR code. This blank space is referred to as the **quiet zone**. You also have the option of modifying the size of this quiet zone by making changes to the `border` parameter within the `.save()` method.

You set the size of the margin around the QR code by specifying an integer value to the `border` argument of the `.save()` method. By default, the size of the quiet zone is four modules on each side.

If you want to completely remove the quiet zone, you can do so by setting `border=0` in the `.save()` method. In the code below, you remove the border from your QR code:

Python

```
# borderless_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "borderless_qrcode.png",
    scale=5,
    border=0,
)
```

You've created a Python script named `borderless_qrcode.py` and set the value of the `border` argument in the `.save()` method to `0`. This will create an image without any blank spaces around the QR code.

To generate a QR code that encodes the text "Hello, World" without a border and with a scaling factor of 5, run `borderless_qrcode.py` from your command line:

Shell

```
$ python borderless_qrcode.py
```

Just like that, you've removed the quiet zone in your QR code. Give it a whirl by scanning your QR code image. You can also scan the QR code image below:



If you want to increase the size of the quiet zone and create a wider border for your QR code, you can do so by increasing the value of `border`. As an example, to create a QR code with a border of ten white modules, you can set `border=10` in the `.save()` method:

Python

```
# wide_border_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "wide_border_qrcode.png",
    scale=5,
    border=10,
)
```

Your Python script named `wide_border_qrcode.py`, in which you set the value of the `border` argument to `10` in the `.save()` method, will create a QR code with a margin of ten light modules.

To generate a 5x5 pixel QR code that encodes the text "Hello, World" with a wider border, run `wide_border_qrcode.py` from your command line:

Shell

```
$ python wide_border_qrcode.py
```


You've just created a PNG file with a QR code of size 5x5 pixels and a border of ten light modules. Check it out:



Since the QR code has a white background, and the color of the quiet zone is white, you might have to enlarge the image to view the wider margin around the QR code. Ideally, you should be able to change the color of the background or quiet zone. In

the next section, you'll learn how to create more colorful QR codes with Python. You'll then be able to change the color of the background, the quiet zone, and the data modules.



 [Remove ads](#)

## Changing the Colors of the QR Code

If you'd like to create QR codes with your favorite colors, then the `.save()` method has optional parameters that you can add to make your codes more colorful. In this section, you'll modify the color of the background, the quiet zone, and the light and dark modules of your QR codes. All you have to do is specify the keyword and a color of choice. You can use the [RGB](#) format, the name of the color, or a [hexadecimal](#) value (`#RRGGBB`).

As an example, if you want to change the color of the background to light blue, then you can add `light="lightblue"` within the `.save()` method:

Python

```
# lightblue_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "lightblue_qrcode.png",
    scale=5,
    light="lightblue",
)
```

You've created a Python script named `lightblue_qrcode.py` and set the value of `lightblue` to the `light` argument within the `.save()` method. This code will replace the white background with a light blue background.

For your reference, you can also replace the color name with the RGB or hexadecimal format. If you want to set the background to light blue in RGB format, then you can replace `light="lightblue"` with `light=(173, 216, 230)`. To use a hexadecimal value, you can replace `light="lightblue"` with `light="#ADD8E6"`.

With that code in place, you can generate a QR code with a background color of light blue by running `lightblue_qrcode.py` from your command line:

Shell

```
$ python lightblue_qrcode.py
```

Great! Now you have a lovely light blue background, and the quiet zone is more visible. To take advantage of your handiwork, scan your QR code image or the QR code image below:





You can also change the colors of all the black modules of the QR code. In order to do so, you can set a value with a color of your choice in the `dark` parameter in the `.save()` method. As an example, the code below will change the colors of the dark modules to dark blue:

Python

```
# darkblue_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "darkblue_qrcode.png",
    scale=5,
    dark="darkblue",
)
```

Your script will set the value of all the black modules to dark blue. To generate this QR code, run `darkblue_qrcode.py` from your command line:

Shell

```
$ python darkblue_qrcode.py
```

In this iteration, your QR code encoding the text "Hello, World" has dark blue modules in place of the traditional black ones:



This time, the quiet zone isn't as visible since the background color is white. In fact, you may have to enlarge the image to see the margins. You can always change the color of the background by specifying a color of your choice in the `light` argument within the `.save()` method.

Another adjustment that you could make to your QR code image is changing the color of the quiet zone by adding a `quiet_zone` parameter in the `.save()` method. Here's an example of an addition you could make to `darkblue_qrcode.py` to change the color of the space around the QR code from white to maroon:

Python

```
# darkblue_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "darkblue_qrcode.png",
    scale=5,
    dark="darkblue",
    quiet_zone="maroon",
)
```

If you run `python darkblue_qrcode.py` from your command line again, then you'll create a QR code with maroon borders:



So far, you've used the `light` and `dark` parameters to change the color of all the white and black modules of the QR code. However, not all of the modules in the QR code house the data that's encoded. If you want to take your QR codes up a level, then you can also change the color of the **data modules**. The data modules are the black and white blocks where the data is actually stored.

To change the colors of the dark data modules of the QR code, you can set a color of your choice for the `data_dark` argument within the `.save()` method. As an example, if you want to change the color of the dark data modules to green, then you can create a Python script called `green_datamodules_qrcode.py` and add `data_dark=green` within the `.save()` method:

Python

```
# green_datadark_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "green_datadark_qrcode.png",
    scale=5,
    light="lightblue",
    dark="darkblue",
    data_dark="green",
)
```

By adding the argument `data_dark` in the `.save()` method and set it to green, you change the colors of the dark data modules to green. You can also change the color of all the dark modules that *don't* contain data to dark blue. To do this, you add `dark="darkblue"`. While you're at it, you go ahead and set the background to light blue by adding `light="lightblue"` within the `.save()` method.

To generate a QR code with green dark data modules, run `green_datadark_qrcode.py` from your command line:

Shell

```
$ python green_datadark_qrcode.py
```

Cool, now you can tell which modules encode data! They stand out in a vibrant green, as you can see below:



You can also add a `data_light` argument with a color of your choice to the `.save()` method to change the colors of the white data modules. To change the colors of the light data modules to light green, you can add `data_light="lightgreen"`:

## Table of Contents

- [Using Python to Generate a Basic QR Code](#)
- [Changing the Size the QR Code](#)
- [Formatting the Border of the QR Code](#)
- [Changing the Colors of the QR Code](#)
- [Rotating the QR Code](#)
- [Creating Animated QR Codes](#)
- [Conclusion](#)

Mark as Completed



Tweet

Share

Email



Python

```
# green_datamodules_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode.save(
    "green_datamodules_qrcode.png",
    scale=5,
    light="lightblue",
    dark="darkblue",
    data_dark="green",
    data_light="lightgreen",
)
```

If you add the argument `data_light` in the `.save()` method and set it to `lightgreen`, this will change the colors of the light data modules to light green. If you run `python green_datamodules_qrcode.py` from your command line again, then you'll create a QR code with dark blue modules and green and light green data modules on a light blue background:



You can now scan your colorful "Hello, World" QR code, or you can scan the QR code image above.

In this section, you've learned how to use the `.save()` method to change the colors of the light and dark parts of the QR codes. Have fun trying different combinations of colors, and you can also refer to the [Segno documentation](#) for a complete list of parameters that you can tweak to create colorful QR codes.

Changing the colors of the background, quiet zone, and modules of your QR code is pretty cool, but you could also rotate your QR codes for a unique creative twist. In the next part of the tutorial, you'll learn how to apply the `.to_pil()` method to rotate your QR codes.



 [Remove ads](#)

## Rotating the QR Code

With Segno, you're also able to manipulate your QR code by rotating it or adding an [animated background](#). If you'd like to create QR codes with advanced graphical operations, then you'll need to install the dependencies `qrcode-artistic` and `pillow`:

Shell

```
$ python -m pip install pillow qrcode-artistic
```

With these dependencies, you'll be able to use two additional methods, `.to_pil()` and `.to_artistic()`, when creating your QR code with `make_qr()`. This will convert your QR code into a [Pillow](#) Image instance, which you can then use to rotate the QR code or replace the static background with an animated instance.

If you'd like to rotate your QR code, then you'll use the `.to_pil()` method and specify the rotation angle in degrees. Below, you encode your trusty "Hello, World" text, but you also rotate the QR code object by 45 degrees:

Python

```
# rotated_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")

qrcode_rotated = qrcode.to_pil().rotate(45)
qrcode_rotated.save("rotated_qrcode.png")
```

You'll have to specify the degree of rotation within the `.to_pil().rotate()` method. The direction of rotation is counterclockwise. By using the `.to_pil()` method and specifying the degree of rotation in the `.rotate()` method, you've written the code to create a black-and-white QR code object rotated by 45 degrees. Finally, you applied the `.save()` method to your `qrcode_rotated` variable.

Now it's time to run `rotated_qrcode.py` from your command line and see what you get:

Shell

```
$ python rotated_qrcode.py
```

It's a little small since you haven't adjust the size (yet), but you can now scan your rotated QR code image, or you can scan the QR code image below:



You may have observed that the QR code is truncated. In particular, the corners of the quiet zone have been cut off. You may have to enlarge the image to notice this. If you'd like to keep the whole image during rotation, then you can add an `expand` argument in the `.to_pil().rotate()` method and set it to `True`. The code below will rotate the QR code by forty-five degrees and expand the canvas to contain the whole code:

Python

```
# rotated_qrcode.py

import segno

qrcode = segno.make("Hello, World")

qrcode_rotated = qrcode.to_pil().rotate(45, expand=True)
qrcode_rotated.save("rotated_qrcode.png")
```

Here, you've specified the degree of orientation and set `expand=True` in the `to_pil().rotate()` method to create a black-and-white QR code object rotated by 45 degrees and enlarged. You then apply the `.save()` method to your `qrcode_rotated` variable and pass the filename with extension.

To generate this QR code, run `python rotated_qrcode.py` from your command line again:



You may want to adjust the size of the rotated QR code as you did in `scaled_qrcode.py`, or you might want to change the background color like you did in `lightblue_qrcode.py`. However, if you want to format a rotated QR code, adding the `scale` or `light` argument to the `.save()` method won't change the size or background color of the QR code. While it looks like before, you're actually calling `.save()` on a PIL object and not a QR code object.

Since you're using the `.to_pil()` method to rotate your QR code, you'll have to add the parameters that resize and change the colors of the QR code within `.to_pil()` instead. Below, you add a scale factor of 5 to the QR code object and change the color of the dark parts of the QR code to green and the light parts to light blue:

Python

```
# formatted_rotated_qrcode.py

import segno

qrcode = segno.make_qr("Hello, World")
qrcode_rotated = qrcode.to_pil(
    scale=5,
    light="lightblue",
    dark="green",
).rotate(45, expand=True)
qrcode_rotated.save("formatted_rotated_qrcode.png")
```

To change the size of the QR code, you set the value in the `scale` argument. You adjust the `light` and `dark` arguments in the `.to_pil()` method to change the color of the light and dark sections of the QR code.

You may have noticed that the rotated QR code has some additional blank space that's not the same as the quiet zone around the QR code. This additional background is created to overlay the rotated QR code. To modify the colors of the dark modules and the additional border, which is also black by default, you can adjust the value of the `dark` argument in the `.to_pil()` method with a color of your choice.

To generate a 5x5 pixel QR code with a light blue background, plus green dark modules and the additional background, run `formatted_rotated_qrcode.py` from your command line:

Shell

```
$ python formatted_rotated_qrcode.py
```

You can scan your formatted and rotated QR code, or you can scan the image below:



Not only can you change the background color of the QR code, but you can also replace it with a moving image. In the final section of this tutorial, you'll use the `.to_artistic()` method to replace a static background with an animated one.

## Creating Animated QR Codes

In the final section of this tutorial, you’re going to create the most beautiful QR code by replacing the static background with an animated image, such as a GIF. This requires a few additional steps, but the results are worth it.

If you’d like to create a QR code with an animated image as the background, then the first thing you need to do is create a Python file called `animated_qrcode.py`. In the example below, you create a QR code object with a YouTube link to the song “[Smells like Teen Spirit](#)” by Nirvana:

Python

```
# animated_qrcode.py

import segno

slts_qrcode = segno.make_qr("https://www.youtube.com/watch?v=hTWKbfoikeg")
```

You’ve just written the code to create a QR code object with the `make_qr()` function. This QR code object encodes the YouTube link for the classic “Smells like Teen Spirit,” or *SLTS* for short. You store this QR code as the variable `slts_qrcode`.

The second step that you’ll have to carry out is to specify the file path of the animated image. If you’d like to use an animated image from a URL, then you’ll need to import and use the `urlopen()` function from the [urllib](#) library. Otherwise, you can skip this step:

Python

```
# animated_qrcode.py

import segno
from urllib.request import urlopen

slts_qrcode = segno.make_qr("https://www.youtube.com/watch?v=hTWKbfoikeg")
nirvana_url = urlopen("https://media.giphy.com/media/LpwBqCorPvZC0/giphy.gif")
```

The `urlopen()` function serves as a way of retrieving URLs, and in this scenario, you’re passing the link to a GIF featuring [Kurt Cobain passionately playing the guitar](#). You store this as a variable called `nirvana_url`. You can skip this step if you use an animated image stored locally, which means you’re not fetching a moving image from a URL.

The final step is to apply the `.to_artistic()` method to your QR code object. This will replace the static background with a moving background. Below, you write the code generate a QR code with modules of 5x5 pixels, with the animated image of Kurt Cobain headbanging as the background:

Python

```
# animated_qrcode.py

import segno
from urllib.request import urlopen

slts_qrcode = segno.make_qr("https://www.youtube.com/watch?v=hTWKbfoikeg")
nirvana_url = urlopen("https://media.giphy.com/media/LpwBqCorPvZC0/giphy.gif")
slts_qrcode.to_artistic(
    background=nirvana_url,
    target="animated_qrcode.gif",
    scale=5,
)
```

In the `.to_artistic()` method, you’ve added a `background` argument with either the file path of your animated image or the variable that stores the image’s URL. You’ve also set the `target` argument with the file path and filename of the QR code image. If you’d like to change the size of the QR code, you can do so by setting the value of the `scale` argument in the `.to_artistic()` method.

When creating artistic QR codes, such as the rotated QR code in `formatted_rotated_qrcode.py` and the QR code that you’re about to generate with an animated background, you don’t use the `.save()` method to format the size or colors of the QR code. If you’d like to format the artistic QR code, then you can assign the parameters within `.to_artistic()`.

To create a GIF file called `animated_qrcode.gif` in the same location where you’re executing your code, you run `animated_qrcode.py` from your command line:

Shell

```
$ python formatted_rotated_qrcode.py
```

By applying the `.to_artistic()` method to your QR code object and setting the value of `background` to the URL of an animated image, you’ve created a QR code that shows Kurt Cobain moshing out:



If you scan the image above or your local QR code image, then this will lead you to the YouTube page for the “Smells like Teen Spirit” music video by Nirvana.

## Conclusion

Now that you have an understanding of how to generate QR codes in Python using the library Segno, you can develop your own beautiful QR Codes. You can make anything from traditional black-and-white QR codes to artistic QR codes with rotated and animated backgrounds.

In this tutorial, you’ve learned how to:

- Use `make_qr()` and `.save()` to create black-and-white QR codes
- Adjust the module **size** and **quiet zone** of your QR code
- Personalize the dark and light modules with your favorite **colors**
- Use the `.to_pil().rotate()` method to rotate your QR codes