## ****How Do QR Codes Work?****

A QR code is a scannable barcode encoded with data. Encoded means converted into a particular form. In the case of QR codes, numeric and alphanumeric characters, bytes, and kanji convert into a unique two-dimensional arrangement of squares. When an optical scanner passes over those squares, it translates their arrangement back into that data’s original form.

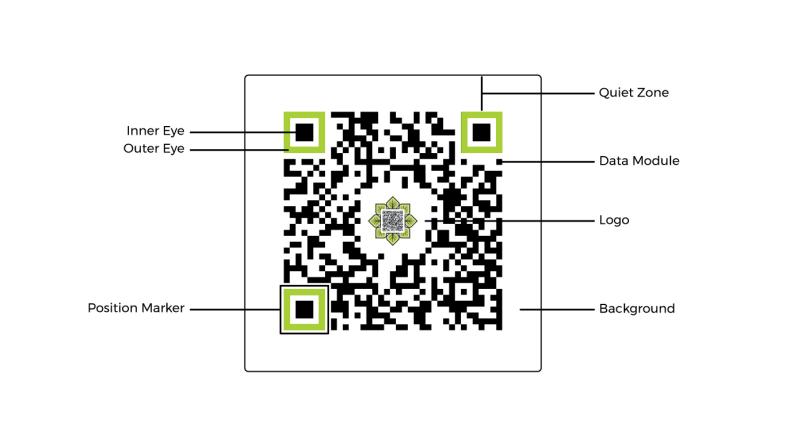
Here’s an example. We create custom, QR code-based [digital restaurant table menus](https://www.sproutqr.com/blog/qr-code-menu" \t "https://www.sproutqr.com/blog/_blank) for hospitality businesses. That’s why we have a [demo digital menu](http://bw-winelist-website-prod.s3-website-us-west-2.amazonaws.com/winelist-demo/" \t "https://www.sproutqr.com/blog/_blank) to show interested parties. To showcase what we can do, we encoded that demo menu’s URL into the QR code below.

A QR code menu example

In the QR code, the arrangement of the squares—or data modules, as they’re called—is actually our URL. It’s just been translated from the alphanumeric string of the URL into a collection of squares. That's how you go from [link to QR code](https://www.sproutqr.com/blog/url-to-qr-code" \t "https://www.sproutqr.com/blog/_blank). A QR code scanner will then translate it back to the original URL.

## ****Parts of a QR Code****

The anatomy of a QR code is mostly relevant to anyone thinking of creating a QR code. Be aware, though, of the [QR code security risks](https://www.sproutqr.com/blog/qr-code-generator-online-risks" \t "https://www.sproutqr.com/blog/_blank) associated with free online services.



The most important parts of a QR code are:

* ****Data module****. This is the standard unit of the QR code. It’s typically a black square set against a white background. Though the colors and contrast can be different, black-on-white is the most optimal when creating a [custom QR code](https://www.sproutqr.com/blog/custom-qr-code" \t "https://www.sproutqr.com/blog/_blank). The arrangement of these black squares, or data modules, is what makes up the majority of a QR code.
* ****Position marker****. There are three position markers on every QR code. Consisting of an inner and outer eye, they allow scanners and cameras to quickly and accurately locate the data modules and the scanning direction.
* ****Quiet zone****. This is the blank area on all sides of the data module matrix that contains all the data modules and position markers. It allows scanners and readers to optically place where the QR code begins and ends.

## ****What Information Is in a QR Code?****

There are three types of information that a QR code stores: size, error correction level, and data type.

### ****Size****

A QR code can be made up of a maximum of 177 rows and 177 columns, which makes for a possible 31,329 data modules. Most QR codes aren’t that big, though.

The size of a QR code corresponds to its version. The smallest a QR code can be is 21 rows by 21 columns, which is version 1. 25x25 is version 2, and on and on. The aforementioned largest QR code possible, 177x177, is version 40.

It's also worth noting that [QR code minimum size](https://www.sproutqr.com/blog/qr-code-minimum-size" \t "https://www.sproutqr.com/blog/_blank) is usually based on scan distance and not data size.

### ****Error Correction Levels****

Encoded in a QR code is one of four [QR code error correction levels](https://www.sproutqr.com/blog/qr-code-wont-scan" \t "https://www.sproutqr.com/blog/_blank). The higher the correction level, the more damage a QR code can sustain while still being scannable. It’s like a stored backup of the QR code. The lower the correction level, the more space left for size and data.

### ****Data Type****

QR codes can store up to 7,089 numeric characters or 2,953 alphanumeric characters. They can also store bytes and kanji, but those are less frequently used. These numbers assume the lowest error correction level.

In practice, this means [QR code uses](https://www.sproutqr.com/blog/qr-code-uses" \t "https://www.sproutqr.com/blog/_blank) include anything that uses numbers, letters, punctuation, and symbols to communicate. Business cards, [QR codes on tables](https://www.sproutqr.com/blog/qr-code-on-table" \t "https://www.sproutqr.com/blog/_blank) in restaurants, authentication, checking into hotels, logging into websites, contactless payments, digital wine lists, [QR code food](https://www.sproutqr.com/blog/qr-code-food" \t "https://www.sproutqr.com/blog/_blank) uses, and more.

For context, the amount of characters a standard one-dimensional barcode can hold is around 20 to 100 characters. This ability of QR codes to store such a large amount of information—and provide it quickly—makes them much more useful tools than standard barcodes. In virtually every industry.

But the more characters you store in a QR code, the bigger, more complicated the QR code has to be, right? Yes and no. That’s the difference between static QR codes and dynamic QR codes.

**General overview of creating** **a QR code**

The following pages of the tutorial will explain the QR code encoding process in detail. Here is a general overview of the process that you can read before moving on to the more detailed steps

#### **Step 1: Data Analysis**

#### A QR code encodes a string of text. The QR standard has four modes for encoding text: numeric, alphanumeric, byte, and Kanji. Each mode encodes the text as a string of bits (1s and 0s), but each mode uses a different method for converting the text into bits, and each encoding method is optimized to encode the data with the shortest possible string of bits. Therefore, your first step should be to perform [data analysis](https://www.thonky.com/qr-code-tutorial/data-analysis) to determine whether your text can be encoded in numeric, alphanumeric, byte, or Kanji mode, then select the most optimal mode for your text.

#### **Step 2: Data Encoding**

#### Now that you have selected the appropriate encoding mode for your text, the next step is to encode the text. The [data encoding](https://www.thonky.com/qr-code-tutorial/data-encoding) section describes this process in detail for each encoding mode. The result of this step is a string of bits that is split up into data codewords that are each 8 bits long.

#### **Step 3: Error Correction Coding**

#### As explained above, QR codes use error correction. This means that after you create the string of data bits that represent your text, you must then use those bits to generate error correction codewords using a process called Reed-Solomon error correction.

#### QR scanners read both the data codewords and the error correction codewords. By comparing the two, the scanner can determine if it read the data correctly, and it can correct errors if it did not read the data correctly. The [error correction coding](https://www.thonky.com/qr-code-tutorial/error-correction-coding) section explains the process of generating error correction codewords in detail. For more information, read Wikipedia's article on [Reed-Solomon error correction](http://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction).

#### **Step 4: Structure Final Message**

#### The data and error correction codewords generated in the previous steps must now be arranged in the proper order. For large QR codes, the data and error correction codewords are generated in blocks, and these blocks must be interleaved according to the QR code specification. This process is explained in the [structure final message](https://www.thonky.com/qr-code-tutorial/structure-final-message) section.

#### **Step 5: Module Placement in Matrix**

#### After generating the data codewords and error correction codewords and arranging them in the correct order, you must place the bits in the QR code matrix. The codewords are arranged in the matrix in a specific way. During this step, you will also place the patterns that are common to all QR codes, such as the boxes on the three corners. This process is explained in detail in the [module placement in matrix](https://www.thonky.com/qr-code-tutorial/module-placement-matrix) section.

#### **Step 6: Data Masking**

#### Certain patterns in the QR code matrix can make it difficult for QR code scanners to correctly read the code. To counteract this, the QR code specification defines eight mask patterns, each of which alters the QR code according to a particular pattern. You must determine which of these mask patterns results in the QR code with the fewest undesirable traits. This is done by evaluating each masked matrix based on four penalty rules. Your final QR code must use the mask pattern that resulted in the lowest penalty score. The masking process is explained in the [data masking](https://www.thonky.com/qr-code-tutorial/data-masking) section.

#### **Step 7: Format and Version Information**

#### The final step is to add format and (if necessary) version information to the QR code by adding pixels in particular areas of the code that were left blank in previous steps. The format pixels identify the error correction level and mask pattern being used in this QR code. The version pixels encode the size of the QR matrix and are only used in larger QR codes. For details about this final step, read the [format and version information](https://www.thonky.com/qr-code-tutorial/format-version-information) section.

Link to the site: <https://www.thonky.com/qr-code-tutorial/format-version-information>

**DATA ANALYSIS**

**- The QR code Modes**

Numeric mode: 0-9;

Alphanumeric mode:

0 0

1 1

2 2

3 3

4 4

5 5

6 6

7 7

8 8

9 9

A 10

B 11

C 12

D 13

E 14

F 15

G 16

H 17

I 18

J 19

K 20

L 21

M 22

N 23

O 24

P 25

Q 26

R 27

S 28

T 29

U 30

V 31

W 32

X 33

Y 34

Z 35

36 (space)

$ 37

% 38

\* 39

+ 40

- 41

. 42

/ 43

: 44

Byte Mode: by default, is for characters from the ISO-8859-1 character set. However, some QR code scanners can automatically detect if UTF-8 is used in byte mode instead.

Kanji mode: is for double-byte characters from the Shift JIS character set. While UTF-8 can encode Kanji characters, it must use three or four bytes to do so. Shift JIS, on the other hand, uses just two bytes to encode each Kanji character, so Kanji mode compresses Kanji characters more efficiently. If the entire input string consists of characters in the double-byte range of Shift JIS, use Kanji mode. It is also possible to use multiple modes within the same QR code, as described later on this page.

**- How to choose the most efficient mode:**

1. If the input string only consist of decimal digits(0 through 9) use numeric mode.
2. If you want to use both numbers and upper case letter (lowercase lettere are not supported) than use alphanumerical mode:
3. If the character that you want is not present in alphanumerical table than you can use byte mode.
4. If you want to use kanji characters than use kanji mode.

**- Mixing Modes and Optimization**

It is possible to use multiple mode in the input text, including the mode indicator before each section of bytes that uses that mode.

**DATA ENCODING**

- Step 1: Choose the Error Correction Level

Before encoding the data, select an error correction level. As mentioned in the introduction, QR codes use Reed-Solomon error correction. This process creates error correction codewords(bytes) based on the encoded data. A QR code reader can use these error correction bytes to determine if it did not read the data correctly, and the error correction codewords can be used to correct those errors. There are four levels of error correction: L, M, Q, H.

L = Recovers 7% of data.

M = Recovers 15% of data.

Q = Recovers 25% of data.

H = Recovers 30% of data.

Be aware higher levels of error correction require more bytes, so the higher the error correction level, the larger the QR code will have to be.

- Step 2: Determine the Smallest Version for the data

The different sizes of QR codes are called versions. There are forty version available. The smallest version si version 1, and is 21 pixels by 21 pixel in size. Each version has a maximum capacity, depending on the mode in use. In addition, the errore correction level restricts the capacity further. The character capacities table lists the capacities of all QR version for a given encoding mode and error correction level.

Link: <https://www.thonky.com/qr-code-tutorial/character-capacities>

Example: HELLO WORLD (lunghezza 11 characters)

For this word i can use version 1 and security Q, because it has 16 alphanumerical characters

- Step 3: Add the Mode Indicator

Each encoding mode has a four-bit mode indicator that identifies it. The encoder data must start with the appropriate that specifies the mode being used for the bits that come after it.

For example, if encoding HELLO WORLD is alphanumeric mode, the mode indicator is 0010.

List:

Numeric Mode = 0001

Alphanumeric Mode = 0010

Byte Mode = 0100

Kanji Mode = 1000

ECI Mode = 0111

- Step 4: Add the Character Count Indicator

The character count indicator is a string of bits that represents the number of characters that are being encoded. The character count indicator must be placed after the mode indicator. Furthermore, the character count indicator must be a certain number of bits long, depending on the QR version.

Count the number of characters in the original input text, then convert that number into binary. The length of the character count indicator depends on the encoding mode and the QR code version that will be in use. To make the binary string the appropriate length, pad it on the left with 0s.

The following lists contain the sizes of the character count indicators for each mode and version. For example, if encoding HELLO WORLD in a version 1 QR code in alphanumeric mode, the character count indicator must be 9 bits long. The character count of HELLO WORLD is 11. In binary, 11 is 1011. Pad it on the left to make it 9 bits long: 000001011. Put this after the mode indicator from step 3 to get the following bit string: 0010 000001011

Versions 1 through 9

Numeric mode: 10 bits

Alphanumeric mode: 9 bits

Byte mode: 8 bits

Japanese mode: 8 bits

Versions 10 through 26

Numeric mode: 12 bits

Alphanumeric mode: 11 bits

Byte mode: 16

Japanese mode: 10 bits

Versions 27 through 40

Numeric mode: 14 bits

Alphanumeric mode: 13 bits

Byte mode: 16 bits

Japanese mode: 12 bits

- Step 3: Encode Using the Selected Mode

The previous page, [data analysis](https://www.thonky.com/qr-code-tutorial/data-analysis), explains how to select the appropriate encoding mode for a given string. The process for each encoding mode is explained on its own page. Click a link below to read about the encoding process for each mode.

[Numeric Mode Encoding](https://www.thonky.com/qr-code-tutorial/numeric-mode-encoding)

[Alphanumeric Mode Encoding](https://www.thonky.com/qr-code-tutorial/alphanumeric-mode-encoding)

[Byte Mode Encoding](https://www.thonky.com/qr-code-tutorial/byte-mode-encoding)

[Kanji Mode Encoding](https://www.thonky.com/qr-code-tutorial/kanji-mode-encoding)

HELLO WORLD is encoded on the [alphanumeric mode encoding](https://www.thonky.com/qr-code-tutorial/alphanumeric-mode-encoding) page. Continuing the HELLO WORLD example, the bit string so far is:

Process of Alphanumeric Mode Encoding:

Break up into pairs

First, break up the string into pairs of characters: HE, LL, O , WO, RL, D

Create a binary number for each pair

For alphanumeric mode, each alphanumeric character is represented by a number. Please refer to the [alphanumeric table](https://www.thonky.com/qr-code-tutorial/alphanumeric-table) to find these numbers. The column on the left shows the alphanumeric character, and the column on the right shows the number that represents it.

For each pair of characters, get the number representation (from the [alphanumeric table](https://www.thonky.com/qr-code-tutorial/alphanumeric-table)) of the first character and multiply it by 45. Then add that number to the number representation of the second character.

For example, the first pair in HELLO WORLD is HE.

H → 17  
E → 14

Following the steps in the previous paragraph, multiply the first number by 45, then add that to the second number:

(45 \* 17) + 14 = 779

Now convert that number into an 11-bit binary string, padding on the left with 0s if necessary.

779 → 01100001011

If you are encoding an odd number of characters, as we are here, take the numeric representation of the final character and convert it into a 6-bit binary string.

HE = 01100001011

LL = 01111000110

O = 10001011100 (we ad space as our second character)

WO = 10110111000

RL = 10011010100

D = 001101 (this is not multiplied by 45 and we did not add space, it 13 in binary)

Mode Indicator = 0010

Character Count Indicator = 000001011

Encoded Data = 01100001011 01111000110 10001011100 10110111000 10011010100 001101

- Step 4: Break Up into 8-bit Codewords and Add Pad Bytes if Necessary

After obtaining a string of bits that consists of the mode indicator, the character count indicator, and the data bits as described in steps 1 through 3 on this page, it may be necessary to add 0s and pad bytes, because the QR code specification requires that the bit string must completely fill the total capacity of the QR code. The following sections explain the process of adding 0s and pad bytes to the bit string.

**Determine the Required Number of Bits for this QR Code**

To determine how many data bits are required for a particular QR code, refer to the [error correction table](https://www.thonky.com/qr-code-tutorial/error-correction-table). Find the version and error correction level that is in use for the QR code being encoded, and find the number in the column that is labeled "Total Number of Data Codewords for this Version and EC Level". Multiply this number by 8 to obtain the total number of data bits required for this version and error correction level.

For example, according to the table, a version 1-Q code has 13 total data codewords. Therefore, the total number of bits required for this QR code is 13 \* 8, or 104 bits.

**Add a Terminator of 0s if Necessary**

If the bit string is shorter than the total number of required bits, a terminator of up to four 0s must be added to the right side of the string. If the bit string is more than four bits shorter than the required number of bits, add four 0s to the end. If the bit string is fewer than four bits shorter, add only the number of 0s that are needed to reach the required number of bits.

For example, if encoding HELLO WORLD in a version 1-Q QR code, the total number of required bits as mentioned in the previous section is 104 bits. The data bit string that is shown in step 3 on this page is 74 bits long. The terminator must only be at most 4 bits long, so add four 0s to the right of the string. The resulting string is still too short to fill the 104 bit capacity, but the QR code specification requires that the terminator be at most four 0s in length. If the string had been 102 bits instead, the terminator would only be 2 bits in length.

Here is the example HELLO WORLD string with terminator added:

Mode Indicator = 0010

Character Count Indicator = 000001011

Encoded Data = 01100001011 01111000110 10001011100 10110111000 10011010100 001101

Terminator: 0000

**Add More 0s to Make the Length a Multiple of 8**

After adding the terminator, if the number of bits in the string is not a multiple of 8, first pad the string on the right with 0s to make the string's length a multiple of 8.

For example, after adding the terminator to the HELLO WORLD string, the length became 78 bits long. This is not a multiple of 8. The bit string is shown here broken up into 8-bit binary bytes:  
00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 010000

There are six bits at the end. Add two 0s to make it an 8-bit binary byte:

00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 010000**00**

**Add Pad Bytes if the String is Still too Short**

If the string is still not long enough to fill the maximum capacity, add the following bytes to the end of the string, repeating until the string has reached the maximum length:  
11101100 00010001

These bytes are equivalent to 236 and 17, respectively. They are specifically required by the QR code specification to be added if the bit string is too short at this stage.

For example, the HELLO WORLD string above is 80 bits long. The required capacity for a 1-Q code, as stated earlier on the page, is 104 bits. The number of bits that must be added to fill the remaining capacity is 104 - 80, or 24. Divide this by 8: 24 /8 = 3. Therefore, three pad bytes must be added to the end of the data string. This is shown below:  
00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 01000000 **11101100 00010001 11101100**