

#  QR Codes

Inhalt

[1 Einleitung 2](#_Toc345855822)

[2 Aufbau von QR Codes 2](#_Toc345855823)

[3 Codierung des Alphabets 5](#_Toc345855824)

[4 Fehlerkorrektur 6](#_Toc345855825)

[Was ist Fehlerkorrektur? 6](#_Toc345855826)

[Fehlerkorrektur bei QR Codes 7](#_Toc345855827)

[Hamming-Code 9](#_Toc345855828)

[5 Maskierung 12](#_Toc345855829)

[6 Gefahren von QR Codes 15](#_Toc345855830)

[Anhang A: ISO-8859-1 16](#_Toc345855831)

[ISO-8859-1 Tabelle 16](#_Toc345855832)

[Umrechnung Hexadezimalsystem <-> Binärsystem 16](#_Toc345855833)

[Anwendung 18](#_Toc345855834)

[Lösungen 19](#_Toc345855835)

[Quellen 23](#_Toc345855836)

# 1 Einleitung

Sie haben bestimmt schon viele QR Codes angetroffen. Diese praktischen Bildchen erlauben es, schnell auf Informationen zuzugreifen und werden zum Beispiel auf Werbeplakaten oder Flyern eingesetzt. Doch wie funktioniert das Prinzip? Was steckt dahinter? Das werden Sie in diesem Skript lernen.

QR Code steht für *Quick Response Code*, weil er schnell ausgelesen und verarbeitet werden kann. Solche Codes werden zum Speichern digitaler Daten auf Papier (oder ähnlichen Medien) benutzt. Die Daten werden also in analoger Form gespeichert. Ein Handy (oder ein anderer Barcodeleser) kann dann diese Daten wieder auslesen, digitalisieren und verarbeiten.

QR Codes wurden erstmals 1994 genutzt, erfunden von der Firma Denso, die eine Tochtergesellschaft von Toyota ist. Diese Firma hat die Codes eingesetzt, um die Produktion von Autobauteilen zu automatisieren und zu kontrollieren. Seit 2000 sind QR Codes ein ISO-Standard. Der Standard wurde einige Male erweitert, weshalb es verschiedene Versionen von QR Codes gibt.

# 2 Aufbau von QR Codes

#### Aufgabe 1:

Betrachten Sie die beiden QR Codes im Titel des Skriptes und dazu diese fünf:

    

1. Welche Gemeinsamkeiten gibt es (rein optisch) zwischen den sieben Codes?
2. Worin unterscheiden sich die Codes?
3. Ihnen sind sicher die Quadrate in drei der vier Ecken aufgefallen. Wozu könnten diese dienen? Warum gibt es sie nicht in allen vier Ecken?

(Die Lösungen zu allen Aufgaben finden Sie gegen Ende des Skriptes, ab Seite 19)

Ein QR Code ist immer ein Quadrat, aufgebaut aus sogenannten *Modulen*. Jedes Modul kann entweder weiss oder schwarz gefärbt sein. Ausserdem sind immer die folgenden Bereiche vorhanden:

 

* Die roten Bereiche (*Position detection patterns*) sind dazu da, die Begrenzung und die Orientierung des Codes anzugeben.
* Der Bereich, der direkt an die roten Bereiche angrenzt, wird leer gelassen, damit die roten Bereiche gut sichtbar sind (hier grün markiert).
* Mit zunehmender Grösse (*Version*) des Codes werden weitere Muster (*Alignment patterns*, hier dunkelblau) hinzugefügt, um die Ausrichtung des Codes besser erkennbar zu machen.

Ein QR Code kann verschiedene Grössen haben. Version 1 besteht aus 21x21 Modulen, Version 2 aus 25x25 Modulen, Version 3 aus 29x29 Modulen usw. Die grösste Version, Version 40, besteht aus 177x177 Modulen. Pro Versionsschritt werden immer 4 Module in der Seitenlänge addiert.

* Die *Timing patterns* (hier gelb) helfen dem Decoder, die Lage der einzelnen Module zu bestimmen. Sie geben sozusagen die Zeilen und Spalten vor. Sie bestehen abwechselnd aus schwarzen und weissen Modulen.
* In den pinken Bereichen wird (für grössere QR Codes) die Versionsnummer angegeben, d.h. es wird angegeben, wie gross der Code ist. Die beiden Bereiche enthalten zweimal dieselbe Information – für den Fall dass der eine Bereich nicht gelesen werden kann.
* Im hellblauen Bereich ist Information über das Format des Codes gespeichert. Diese Informationen geben wichtige Details an, z.B. welche Maske verwendet wurde und welche Fehlerkorrektur angewandt wurde. Zu diesen Details folgt später mehr.

Wie die Versions-Information ist auch die Format-Information doppelt vorhanden. Einmal im Gegenuhrzeigersinn um das obere linke *Position detection pattern*, das zweite Mal zuerst links unten aufwärts, dann rechts oben von links nach rechts.

Der Rest des QR-Codes besteht aus den Daten, also z.B. dem Text, welche den eigentlichen Inhalt ausmachen. Das "Gerüst" wird wie folgt befüllt:



Quelle: http://upload.wikimedia.org/wikipedia/commons/thumb/1/1e/ QR\_Code\_Unmasked.svg/528px-QR\_Code\_Unmasked.svg.png

Beginnend bei der unteren rechten Ecke werden die Daten (eine Folge von 0 und 1) eingetragen. Zuerst kommen (in grau) die eigentlichen Daten, danach (in grün) die Bits, welche für die Fehlerkorrektur zuständig sind. Falls am Schluss noch Platz übrig ist, wird dieser einfach leer gelassen.

#### Aufgabe 2:

Hier ist ein QR Code, der noch nicht fertig gestellt ist. Füllen Sie die hellgelb markierten Bereiche richtig auf. Die fehlenden Bits für den Anfang in der rechten unteren Ecke lauten:

1000 0011 1000 0110 1000 0101 1100 0001 1010 0001 1101 0110



# 3 Codierung des Alphabets

Am Anfang jedes QR Codes steht eine Information (d.h. ein Text / eine URL / eine Zahl), die dargestellt werden soll. Wie wird diese Information in Nullen und Einsen umgewandelt, so dass man dies im QR Code eintragen kann?

Abhängig davon, welche Zeichen im Text vorkommen, kann man den Text mit verschiedenen Zeichensätzen codieren:

* **Zahlen (0-9)**Kommen nur Ziffern vor, kann ein QR Code theoretisch bis 7089 Zeichen speichern
* **Alphanumerisch (0-9A-Z $%\*+-./:)**Kommen nur Grossbuchstaben, Zahlen und 9 vorgegebene Sonderzeichen vor, können bis zu 4296 Zeichen gespeichert werden
* **ISO-8859-1**Wenn der Text nur aus Zeichen besteht, die in ISO-8859-1 vorkommen (das sind unter anderem Gross- und Kleinbuchstaben, Ziffern, viele Satzzeichen und kombinierte Buchstaben für den westeuropäischen Sprachraum), können bis zu 2953 Zeichen codiert werden
* **KANJI**Wenn der Text nur aus KANJI (japanische Schriftzeichen) besteht, können maximal 1817 Zeichen gespeichert werden.

Nachdem der passende Zeichensatz bestimmt ist, werden die folgenden Informationen in die Bitfolge geschrieben:

1. **Die Kennnummer des Zeichensatzes**

Jeder der vier Zeichensätze hat eine eindeutige Nummer, sozusagen der Name des Zeichensatzes

1. **Die Anzahl der Zeichen, die der Text hat**

Damit man weiss, wie viel Text jetzt kommt

1. **Der Text selbst**
2. **Die Ende-Kennung; sie ist immer 0000**

Wenn man 0000 liest, heisst das, dass der Text jetzt fertig ist

#### Beispiel "Märchenbuch":

1. **Kennnummer:**

Um den Text "Märchenbuch" zu codieren, wird zuerst der passende Zeichensatz ausgewählt. In diesem Fall ist das ISO-8859-1. Dieser Zeichensatz hat die Kennnummer **0100**.

1. **Anzahl Zeichen:**

Im zweiten Schritt wird gezählt, wie viele Zeichen der Text enthält. In diesem Fall sind es 11 Zeichen. Bei kleinen QR-Codes wird diese Zahl mit 8 Bit codiert - bei grösseren QR-Codes würde die Zahl mit 16 Bit codiert werden. Die Zahl 11 soll also im Binärsystem geschrieben werden und zwar mit 8 Stellen (8 Bits). Das ergibt **0000 1011.**

1. **Der Text selbst:**

Anschliessend werden die einzelnen Zeichen codiert. Für ISO-8859-1 ist im Anhang A auf Seite 16 angegeben, wie diese Codierung erfolgt. Das "M" wird zu **0100 1101**, das "ä" zu **1110 0100** und so weiter.

1. **Ende-Kennung:**

Den Abschluss bildet die Ende-Kennung. Sie ist immer **0000**, ausser wenn nicht mehr genug Platz ist. Dann wird sie abgekürzt.

Insgesamt wird aus dem Text "Märchenbuch" somit diese Bitfolge:

* 0100
* 0000 1011
* 0100 1101 1110 0100 0111 0010 0110 0011 0110 1000 0110 0101 0110 1110 0110 0010 0111 0101 0110 0011 0110 1000
* 0000

#### Aufgabe 3:

Codieren Sie den Text "Kanti Wil" analog zum Beispiel Märchenbuch. Verwenden Sie dazu auch ISO-8859-1. Hinweis: Den Leerschlag finden Sie in der Tabelle unter "NBSP"

# 4 Fehlerkorrektur

Sie wissen bereits, wie aus gegebenen Informationen eine Bitfolge hergestellt wird und wie diese danach in einen QR Code eingetragen wird. Der nächste Schritt ist die Fehlerkorrektur.

## Was ist Fehlerkorrektur?

#### Aufgabe 4:

Lassen Sie sich vom Lehrer / der Lehrerin den Zaubertrick mit den Kärtchen demonstrieren.

1. Wie funktioniert der Trick?
2. Funktioniert er auch noch, wenn zwei Kärtchen umgedreht werden?

#### Was steckt hinter dem Trick?

Die zwei zusätzlichen Zeilen und Spalten dienen der "Fehlerkorrektur". Mit diesen zwei zusätzlichen Reihen wird sichergestellt, dass im ursprünglichen Bild eine gewisse Bedingung erfüllt ist (in jeder Zeile und Spalte eine gerade Anzahl schwarze Kärtchen).

Sobald im geänderten Bild diese Bedingung an einer Stelle nicht mehr erfüllt ist, muss ein "Fehler", d.h. eine Veränderung vorliegen. Weil man sehen kann, in welcher Zeile und in welcher Spalte der Fehler liegt, lässt sich das gedrehte Kärtchen eindeutig bestimmen. Die Fehlerkorrektur sorgt also nicht nur dafür, dass man feststellen kann, dass irgendetwas falsch ist, sondern sie erlaubt es auch, den Fehler genau zu lokalisieren und damit zu korrigieren.

Wenn Bitfolgen von Nullen und Einsen übertragen werden sollen, dann passieren immer ein paar Übertragungsfehler. Deshalb werden in solchen Fällen zusätzliche Fehlerkorrekturbits angehängt oder eingefügt, um gewisse Bedingungen zu erfüllen, ähnlich wie bei den Kärtchen.

Wird die Bitfolge übertragen, kann der Empfänger jederzeit überprüfen, ob die Bedingungen erfüllt sind. Wenn nicht, ist bei der Übertragung etwas schief gelaufen. Mit Hilfe der Fehlerkorrektur kann der Empfänger sogar genau bestimmen, welches Bit falsch empfangen worden ist und es berichtigen.

## Fehlerkorrektur bei QR Codes

Bei QR Codes sind solche fehlerkorrigierenden Codes sehr wichtig. Da sich die QR Codes meistens auf Papier befinden, können Verschmutzungen vorkommen oder es kann sein, dass das Papier an einer Stelle zerknittert ist. Dann wird der QR Code vom Barcodeleser falsch gelesen und es könnte zu Fehlern kommen.

#### Aufgabe 5:

Die folgenden vier QR Codes enthalten alle dieselbe Information. Sie besitzen allerdings verschiedene "Fehlerkorrekturlevels". Das Fehlerkorrekturlevel gibt an, wie viel Fehlerkorrektur zusätzlich zu der eigentlichen Information im QR Code vorhanden ist.

1. Falls Sie auf Ihrem Handy noch keine App zum Lesen von QR Codes haben, dann laden Sie sich jetzt eine herunter. Es gibt viele solche Gratis-Apps, z.B. "Barcode Scanner" für Android oder "NeoReader" für iPhone. (Falls Sie mit dem NeoReader arbeiten, schalten Sie in den Einstellungen "Scan bestätigen" ein).
2. Untersuchen Sie die unten stehenden Codes. Wie viel Prozent des Codes kann man abdecken / unkenntlich machen / zerstören / ..., so dass die App die im Code enthaltenen Daten noch lesen kann? Schätzen Sie für jeden Code eine Prozentzahl. Sie bekommen die Codes auch auf einem separaten Blatt, etwas grösser, damit Sie gut damit arbeiten können.

|  |  |  |
| --- | --- | --- |
| QR Code | Fehlerkorrekturlevel | Toleriert ... % Fehler |
|  | **M** |  |

|  |  |  |
| --- | --- | --- |
| QR Code | Fehlerkorrekturlevel | Toleriert ... % Fehler |
|  | **H** |  |
|  | **L** |  |
|  | **Q** |  |

Bei Level H kann bis zu 30% des QR Codes verschmutzt oder abgedeckt sein und die Information wird trotzdem noch erkannt. Dank dieser Tatsache können so genannte *Design QR Codes*, die mit Logos oder Bildern versehen sind, immer noch gelesen werden.

Beispiele für Design QR Codes (von http://www.qrcode-generator.de/design):

   

Neben den "verdeckten" Bereichen sind bei Design QR Codes zusätzlich oft die Ecken abgerundet und es werden verschiedene Farben verwendet. Allerdings können gewisse Barcodescanner Probleme mit dem Lesen bekommen, wenn der Code zu fest "verschönert" wird.

Doch wie funktioniert diese Fehlerkorrektur bei QR Codes? Die Korrektur basiert auf dem Solomon-Reed Algorithmus, der leider sehr fortgeschrittene mathematische Kenntnisse erfordert, wenn man ihn verstehen will. Weil dies den Rahmen des Ergänzungsfachs sprengen würde, schauen wir uns ein einfacheres Verfahren zu Fehlerkorrektur an, das zwar nicht bei QR Codes aber sonst in verschiedenen Bereichen der Informatik verwendet wird.

## Hamming-Code

Der Hamming-Code ist ein Verfahren zur Fehlerkorrektur, das von Richard Hamming (1915-1998) entwickelt worden ist. Mit ihm können Bitfolgen so modifiziert werden, dass bei einer Übertragung ein falsches Bit erkannt und korrigiert werden kann.

#### Hamming –Code anhand eines Beispiels:

**Codieren:**

Daten, die codiert werden sollen:

0110100010011

####  Platz für Kontrollbits machen

An allen Stellen, die Zweierpotenzen sind, wird Platz für ein Kontrollbit gemacht:

\_ \_0\_110\_1000100\_11

(Hier sind an den Stellen 1, 2, 4, 8, 16 Kontrollbits)

#### Erstes Kontrollbit

Für das erste Kontrollbit wird immer ein Bit der Folge angeschaut, dann eines ausgelassen, dann wieder eines angeschaut... Beginn ist beim ersten Kontrollbit (d.h. Bit 1).

**\_** \_**0**\_**1**1**0**\_**1**0**0**0**1**0**0**\_**1**1

 In den markierten Bits muss eine gerade Anzahl 1 vorkommen. Hier haben wir vier 1 in den markierten Bits, d.h. das Kontrollbit muss eine 0 sein. Somit haben wir:

**0**\_0\_110\_1000100\_11

#### Zweites Kontrollbit

Für das zweite Kontrollbit werden immer zwei Bits angeschaut, dann zwei ausgelassen usw. Beginn ist beim zweiten Kontrollbit (d.h. Bit 2).

0**\_0**\_1**10**\_1**00**01**00**\_1**1**

In den markierten Bits muss eine gerade Anzahl 1 vorkommen. Hier haben wir zwei 1 in den markierten Bits, d.h. das Kontrollbit muss eine 0 sein. Somit haben wir:

0**0**0\_110\_1000100\_11

#### Drittes Kontrollbit

Beginn ist bei Bit 4 (drittes Kontrollbit) und es werden immer 4 Bits angeschaut, dann 4 ausgelassen usw.

000**\_110**\_100**0100**\_11

In den markierten Bits muss eine gerade Anzahl 1 vorkommen. Hier haben wir drei 1 in den markierten Bits, d.h. das Kontrollbit muss eine 1 sein. Somit haben wir:

000**1**110\_1000100\_11

#### Viertes Kontrollbit

Beginn ist bei Bit 8 (viertes Kontrollbit) und es werden immer 8 Bits angeschaut, dann 8 ausgelassen usw.

0001110**\_1000100**\_11

In den markierten Bits muss eine gerade Anzahl 1 vorkommen. Hier haben wir zwei 1 in den markierten Bits, d.h. das Kontrollbit muss eine 0 sein. Somit haben wir:

0001110**0**1000100\_11

#### Fünftes Kontrollbit

Beginn ist bei Bit 16 (fünftes Kontrollbit) und es werden immer 16 Bits angeschaut, dann 16 ausgelassen usw.

000111001000100**\_11**

In den markierten Bits muss eine gerade Anzahl 1 vorkommen. Hier haben wir zwei 1 in den markierten Datenbits, d.h. das Kontrollbit muss eine 0 sein. Somit haben wir:

000111001000100**0**11

Et voilà: Die Codierung ist fertig!

|  |
| --- |
| Merke:Beim Kontrollbit an der Stelle n: Beginne bei Bit n und markiere n Bits, lasse n Bits weg usw. Die Anzahl 1 muss immer gerade sein. |

**Überprüfen:**

Zu überprüfende Bitfolge:

010010001000101101

####  Erstes Kontrollbit überprüfen

Für das erste Kontrollbit wird immer ein Bit der Daten angeschaut, dann eines ausgelassen, dann wieder eines angeschaut... Beginn ist beim ersten Kontrollbit (d.h. Bit 1).

**0**1**0**0**1**0**0**0**1**0**0**0**1**0**1**1**0**1

 Ziel: Eine gerade Anzahl 1. Hier haben wir vier davon, also ist dieses Kontrollbit ok.

#### Zweites Kontrollbit überprüfen

Für das zweite Kontrollbit werden immer zwei Bits angeschaut, dann zwei ausgelassen usw. Beginn ist beim zweiten Kontrollbit (d.h. Bit 2).

0**10**01**00**01**00**01**01**10**1**

 Ziel: Eine gerade Anzahl 1. Hier haben wir drei davon, also ist dieses Kontrollbit nicht ok.

#### Drittes Kontrollbit überprüfen

Beginn ist bei Bit 4 (drittes Kontrollbit) und es werden immer 4 Bits angeschaut, dann 4 ausgelassen usw.

010**0100**0100**0101**101

 Ziel: Eine gerade Anzahl 1. Hier haben wir drei davon, also ist dieses Kontrollbit nicht ok.

#### Viertes Kontrollbit überprüfen

Beginn ist bei Bit 8 (viertes Kontrollbit) und es werden immer 8 Bits angeschaut, dann 8 ausgelassen usw.

0100100**01000101**101

 Ziel: Eine gerade Anzahl 1. Hier haben wir drei davon, also ist dieses Kontrollbit nicht ok.

#### Fünftes Kontrollbit überprüfen

Beginn ist bei Bit 16 (fünftes Kontrollbit) und es werden immer 16 Bits angeschaut, dann 16 ausgelassen usw.

010010001000101**101**

 Ziel: Eine gerade Anzahl 1. Hier haben wir zwei davon, also ist dieses Kontrollbit ok.

#### Falsche Kontrollbits addieren

Die falschen Kontrollbits sind die Bits 2, 4 und 8. Die Summe davon ist 2 + 4 + 8 = 14, d.h. bei der vorliegenden Zeichenfolge ist das 14. Bit falsch. Mathe-Magie! :-)

0100100010001**0**1101 falsches Bit

0100100010001**1**1101 so wäre es richtig

|  |
| --- |
| Merke:Beim Kontrollbit an der Stelle n: Beginne bei Bit n und markiere n Bits, lasse n Bits weg usw. Die Anzahl 1 muss gerade sein, sonst ist das Kontrollbit falsch.Addiere alle falschen Kontrollbits um das gekippte Bit zu erhalten. |

#### Aufgabe 6:

1. Codieren Sie die Bitfolge 1001011010111001 mit dem Hamming-Code.
2. Sie empfangen die Folge 11100001110010101010. Wurde sie richtig übertragen? Wenn nein: Welches Bit ist falsch?

#### Wie funktioniert der Hamming-Code?

Das erste Kontrollbit schaut auf die Stellen 1, 3, 5, 7, 9, ... Schreibt man diese Zahlen um als Summen von Zweierpotenzen, so sind es die Stellen 1, 2+1, 4+1, 4+2+1, 8+1, ... also alle Stellen, die in dieser Summendarstellung "+1" enthalten.

#### Aufgabe 7:

Füllen Sie die Tabelle aus. Geben Sie jeweils an, welche Stellen das Kontrollbit überprüft und wie die Stellennummer als Summe von Zweierpotenzen geschrieben wird.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Kontrollbit | 2. Kontrollbit | 3. Kontrollbit | 4. Kontrollbit |
| Stelle | Summe | Stelle | Summe | Stelle | Summe | Stelle | Summe |
| 1 | 1 |  |  |  |  |  |  |
| 3 | 2+1 |  |  |  |  |  |  |
| 5 | 4+1 |  |  |  |  |  |  |
| 7 | 4+2+1 |  |  |  |  |  |  |
| 9 | 8+1 |  |  |  |  |  |  |
| Alle Stellen, die folgendes enthalten:+1 |  |  |  |

Wenn das erste Kontrollbit (Stelle 1) also falsch ist, muss der Fehler an einer Stelle sein, deren Summe "+1" enthält. Wenn das zweite Kontrollbit (Stelle 2) falsch ist, muss der Fehler an einer Stelle sein, die "+2" enthält, und so weiter.

Erhält man als falsche Kontrollbits die Stellen 1, 2 und 8, so muss die fehlerhafte Stellennummer "+1", "+2" und "+8" enthalten – und sonst nichts, denn sonst wäre ein weiteres Kontrollbit falsch! Die falsche Stelle ist also 1 + 2 + 8 = 11.

# 5 Maskierung

Nun wissen Sie, wie man den gesamten QR Code mit den schwarzen und weissen Modulen ausfüllt. Den Anfang machen die Datenbits, danach folgen Fehlerkorrekturbits. Was jetzt noch passieren kann, ist dass es sehr grosse zusammenhängende schwarze oder weisse Flächen gibt.

Wenn Sie sich fertige QR Codes anschauen, dann stellen Sie fest, dass es selten grosse einfarbige Flächen gibt – meist sind die Codes sehr "unruhig". Beim letzten Schritt der QR Code-Erstellung, der sogenannten Maskierung, wird sichergestellt, dass genau dies der Fall ist.

#### Aufgabe 8:

Warum macht man diesen letzten Schritt? Was könnte theoretisch passieren, wenn ein QR Code einfach mit den erhaltenen Daten und Fehlerkorrekturbits gefüllt wird?

Bei der Maskierung wird eine Art "Folie" über den QR Code gelegt. Auf der Folie sind Bereiche markiert, in welchen die Module des QR Codes umgedreht werden sollen, d.h. aus schwarz wird weiss und umgekehrt.

#### Beispiel:

|  |  |  |  |
| --- | --- | --- | --- |
| Maske1.gif | Maske2.gif | Maske3.gif | Maske4.gif |
| In diesem QR Codes gibt es grosse einfarbige Flächen | Diese Maske wird über den Code gelegt. Ein weisses Feld bedeutet, dass der Code so belas­sen wird, gelb heisst, die Farbe wird an dieser Stelle geändert | Der ursprüngliche Code plus die darübergelegte Maske | Im Endresultat gibt es viel weniger einfarbige Flächen |

#### Aufgabe 9:

Maskieren Sie den gegebenen QR Code mit der Maske:

|  |  |  |
| --- | --- | --- |
| **Code** | **Maske** | **Lösung** |
| Maske_Aufg1.gif | Maske_Aufg2.gif | Maske_Aufg4.gif |

Für QR Codes gibt es acht verschiedene Masken:



Quelle: http://research.swtch.com/qart

Die Masken bestehen jeweils aus einem Muster, das immer wiederholt wird. Man sieht bei den obigen Masken, dass wichtige Bereiche wie die *Position Detection Patterns* in den Ecken, die *Alignment Patterns* und natürlich auch Versions- und Formatinformationen nicht maskiert werden. Denn bei diesen Informationen ist es zwingend, dass sie unverändert und sofort lesbar sind.

Wenn ein QR Code fertig ausgefüllt ist, werden alle dieser acht Masken darauf angewendet. Dies ergibt acht verschiedene fertige QR Codes. Von diesen acht wird derjenige ausgewählt, der am wenigsten "störende Eigenschaften" aufweist. Diese "störenden Eigenschaften" sind im Standard für QR Codes festgelegt. Für die acht möglichen Codes wird anhand von vorgegebenen Kriterien ausgerechnet, wie "schlecht" der Code ist. Derjenige QR Code mit den wenigsten "Strafpunkten" wird als Endresultat gewählt.



http://qrbcn.com/imatgesbloc/Three\_QR\_Code.pdf

Damit ist der QR Code endgültig fertig und kann von Anwendern auf der ganzen Welt gescannt werden.

# 6 Gefahren von QR Codes

QR Codes sind praktisch. Man kann damit schnell auf eine Webseite verlinken, Nachrichten austauschen, Kontaktdaten verschicken etc. Doch wie so vieles heutzutage bieten sie auch Angriffspunkte für Hacker.

#### Aufgabe 10:

1. Recherchieren Sie im Internet. Welche Gefahren können von QR Codes ausgehen?
2. Was kann ein Hacker im schlimmsten Fall mit den Daten auf Ihrem Handy anfangen? Was wäre für Sie das Schlimmste?
3. Für wie gefährlich halten Sie QR Codes? Wie realistisch ist ein Angriff?
4. Wie kann man sich gegen die Gefahren schützen?

# Anhang A: ISO-8859-1

## ISO-8859-1 Tabelle

###

Quelle: http://de.wikipedia.org/wiki/ISO\_8859-1

## Umrechnung Hexadezimalsystem <-> Binärsystem

###

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 0123456789101112131415 | 0123456789ABCDEF | 000001002003004005006007010011012013014015016017 | 00000000000000010000001000000011000001000000010100000110000001110000100000001001000010100000101100001100000011010000111000001111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 16171819202122232425262728293031 | 101112131415161718191A1B1C1D1E1F | 020021022023024025026027030031032033034035036037 | 00010000000100010001001000010011000101000001010100010110000101110001100000011001000110100001101100011100000111010001111000011111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 32333435363738394041424344454647 | 202122232425262728292A2B2C2D2E2F | 040041042043044045046047050051052053054055056057 | 00100000001000010010001000100011001001000010010100100110001001110010100000101001001010100010101100101100001011010010111000101111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 48495051525354555657585960616263 | 303132333435363738393A3B3C3D3E3F | 060061062063064065066067070071072073074075076077 | 00110000001100010011001000110011001101000011010100110110001101110011100000111001001110100011101100111100001111010011111000111111 |

 |
|

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 64656667686970717273747576777879 | 404142434445464748494A4B4C4D4E4F | 100101102103104105106107110111112113114115116117 | 01000000010000010100001001000011010001000100010101000110010001110100100001001001010010100100101101001100010011010100111001001111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 80818283848586878889909192939495 | 505152535455565758595A5B5C5D5E5F | 120121122123124125126127130131132133134135136137 | 01010000010100010101001001010011010101000101010101010110010101110101100001011001010110100101101101011100010111010101111001011111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 96979899100101102103104105106107108109110111 | 606162636465666768696A6B6C6D6E6F | 140141142143144145146147150151152153154155156157 | 01100000011000010110001001100011011001000110010101100110011001110110100001101001011010100110101101101100011011010110111001101111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 112113114115116117118119120121122123124125126127 | 707172737475767778797A7B7C7D7E7F | 160161162163164165166167170171172173174175176177 | 01110000011100010111001001110011011101000111010101110110011101110111100001111001011110100111101101111100011111010111111001111111 |

 |
|

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 128129130131132133134135136137138139140141142143 | 808182838485868788898A8B8C8D8E8F | 200201202203204205206207210211212213214215216217 | 10000000100000011000001010000011100001001000010110000110100001111000100010001001100010101000101110001100100011011000111010001111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 144145146147148149150151152153154155156157158159 | 909192939495969798999A9B9C9D9E9F | 220221222223224225226227230231232233234235236237 | 10010000100100011001001010010011100101001001010110010110100101111001100010011001100110101001101110011100100111011001111010011111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 160161162163164165166167168169170171172173174175 | A0A1A2A3A4A5A6A7A8A9AAABACADAEAF | 240241242243244245246247250251252253254255256257 | 10100000101000011010001010100011101001001010010110100110101001111010100010101001101010101010101110101100101011011010111010101111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 176177178179180181182183184185186187188189190191 | B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF | 260261262263264265266267270271272273274275276277 | 10110000101100011011001010110011101101001011010110110110101101111011100010111001101110101011101110111100101111011011111010111111 |

 |
|

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 192193194195196197198199200201202203204205206207 | C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF | 300301302303304305306307310311312313314315316317 | 11000000110000011100001011000011110001001100010111000110110001111100100011001001110010101100101111001100110011011100111011001111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 208209210211212213214215216217218219220221222223 | D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF | 320321322323324325326327330331332333334335336337 | 11010000110100011101001011010011110101001101010111010110110101111101100011011001110110101101101111011100110111011101111011011111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 224225226227228229230231232233234235236237238239 | E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF | 340341342343344345346347350351352353354355356357 | 11100000111000011110001011100011111001001110010111100110111001111110100011101001111010101110101111101100111011011110111011101111 |

 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec** | **Hex** | **Oct** | **Binaire** |
|  |
| 240241242243244245246247248249250251252253254255 | F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF | 360361362363364365366367370371372373374375376377 | 11110000111100011111001011110011111101001111010111110110111101111111100011111001111110101111101111111100111111011111111011111111 |

 |

## Anwendung

Gesucht: Binäre Bitfolge, die das Zeichen "B" in ISO 8859-1 codiert.

Vorgehen:

* Das Zeichen in der ISO-Tabelle suchen:

Der Code dafür ist 42 (hexadezimal)

* Den hexadezimalen Code mit der Umrechnungstabelle in Binärcode umwandeln:

42 Hex entspricht 01000010 Binär

Die Codierung von B ist also 01000010.



#  Lösungen

#### Aufgabe 1:

1. Gleiches Muster in drei Ecken (oben links und rechts, unten links), bei grösseren Codes eine kleine Version dieses Quadrates in regelmässigen Abständen.
2. In der Grösse (Anzahl "Module"), bei grösseren Codes gibt es mehr von den kleinen Quadraten
3. Die so genannten "Position Detection Patterns" sind dazu da, die Begrenzung des QR Codes zu erfassen. Es gibt sie nur in drei Ecken, damit die Orientierung des Codes immer klar ist, auch wenn man ihn auf dem Kopf oder gedreht einscannt.

#### Aufgabe 2:



#### Aufgabe 3:

* 0100
* 0000 1000
* 0100 1011 0110 0001 0110 1110 0111 0100 0110 1001 1010 0000 0101 0111 0110 1001 0110 1100
* 0000

#### Aufgabe 4:

1. Der Lehrer hat sichergestellt, dass in jeder Zeile und in jeder Spalte eine gerade Anzahl x vorkommt. Wird ein Kärtchen vertauscht, dann sorgt das dafür, dass in der betreffenden Zeile und der betreffenden Spalte die Anzahl x ungerade wird. Damit kann nach der Vertauschung schnell festgestellt werden, welches das umgedrehte Kärtchen ist.
2. Ja, wenn zwei Kärtchen aus verschiedenen Zeilen und Spalten umgedreht werden. Wenn es zwei Kärtchen aus der gleichen Zeile sind, dann kann man im Nachhinein nur feststellen, dass es a) zwei Kärtchen aus einer Zeile waren und b) in welchen Spalten sie liegen. Die betreffende Zeile lässt sich aber nicht angeben.

#### Aufgabe 5:

|  |  |
| --- | --- |
| Fehlerkorrekturlevel | Toleriert ... % Fehler |
| M | 15% |
| H | 30% |
| L | 7% |
| Q | 25% |

#### Aufgabe 6:

1. 101000100110101111001
2. Die Kontrollbits 1, 2 und 8 sind falsch, d.h. das 11. Bit müsste eine 1 sein statt einer 0.

#### Aufgabe 7:

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Kontrollbit | 2. Kontrollbit | 3. Kontrollbit | 4. Kontrollbit |
| Stelle | Summe | Stelle | Summe | Stelle | Summe | Stelle | Summe |
| 1 | 1 | 2 | 2 | 4 | 4 | 8 | 8 |
| 3 | 2+1 | 3 | 2+1 | 5 | 4+1 | 9 | 8+1 |
| 5 | 4+1 | 6 | 4+2 | 6 | 4+2 | 10 | 8+2 |
| 7 | 4+2+1 | 7 | 4+2+1 | 7 | 4+2+1 | 11 | 8+2+1 |
| 9 | 8+1 | 10 | 8+2 | 12 | 8+4 | 12 | 8+4 |
| Alle Stellen, die folgendes enthalten:+1 | Alle Stellen, die folgendes enthalten:+2 | Alle Stellen, die folgendes enthalten:+4 | Alle Stellen, die folgendes enthalten:+8 |

#### Aufgabe 8:

Die Maskierung sorgt dafür, dass bestimmte Muster, wie das Position Detection Pattern, in Datenmodulen unterdrückt wird. Dadurch wird eine schnelle Erkennung des Symbols garantiert.

Es könnte sonst passieren, dass die Daten- oder Fehlerkorrekturbits im Code per Zufall genau so angeordnet sind, dass fälschlicherweise ein Muster entsteht, das dem Alignment Pattern oder dem Position Detection Pattern entspricht. Dann hätte ein Decoder Mühe, den Code richtig zu lesen.

#### Aufgabe 9:

|  |  |  |
| --- | --- | --- |
| **Code** | **Maske** | **Lösung** |
| Maske_Aufg1.gif | Maske_Aufg2.gif | Maske_Aufg4.gif |

#### Aufgabe 10:

1. Wenn man mit seinem Handy einen QR Code scannt, kann dieser einen Link auf eine Webseite enthalten. Je nachdem, wie die App eingestellt ist, führt sie einen eventuell direkt auf die Webseite – ohne Nachfrage. Theoretisch kann von dieser Website aus Schadsoftware auf das Handy geladen werden, ohne dass der Benutzer es merkt. Diese läuft dann in Zukunft im Hintergrund mit und kann Daten aufzeichnen etc.
2. Mögliche Szenarien sind
	* Auslesen von Passwörtern
	* Daten werden hoch- und runtergeladen, was zu einer höheren Handyrechnung führen kann
	* Teure Telefonnummern können angerufen werden
	* Die Kontakte können ausgelesen werden. Einige Leute sind ziemlich naiv und speichern ihre Passwörter dort...
	* "Der Feind" kann GPS-Daten auslesen, d.h. er weiss, wo man sich befindet
	* Die Fotos auf dem Handy können ausgelesen werden
	* Der gescannte Link kann auf eine Phishing-Seite führen. Die sieht vielleicht aus wie eine ganz normale Seite (paypal, ebay,...), ist aber ein Fake und nur dazu da, Passwörter auszulesen.
	* Wenn es ein Firmenhandy ist, dann freut sich der Hacker natürlich umso mehr, wenn er Zugangsdaten und Passwörter bekommt...
	* ...
3. Wenn ein russischer Hacker mich damit angreifen will, muss er zuerst in die Schweiz kommen und dort einen QR Code an irgendeine Bahnhofswand oder so kleben (oder einfacher: übers Internet). Diese muss ich dann scannen und ihr blind vertrauen. Dann muss ich auch noch genau das richtige Betriebssystem auf dem Handy haben, damit der Angriff erfolgreich ist.

Es gibt Angriffe in dieser Form, sie sind auch schon erfolgreich gewesen, aber es gibt deutlich gefährlichere Dinge, die man im Internet anstellen kann.

* + Man sollte seine Handy-App so einstellen, dass sie nicht direkt auf eine Website führt
	+ Sehr grosse QR Codes sollten einem suspekt vorkommen.
	+ Bei Android-Smartphones: Besser keine Apps via QR Code installieren! Das könnte irgendeine suspekte App sein!
	+ Auch auf Werbeplakaten von bekannten Firmen finden sich gefährliche QR Codes. Je nach Plakat ist es nämlich sehr einfach, den vorhandenen Code mit einem anderen zu überkleben.

# Quellen

* <http://www.qrcode.com>, Stand 20.12.12
* Einführung in QR Codes von Martin Stoev
<http://martinstoev.de/public/articles/qrcode/ausarbeitung/index.html>, Stand 20.12.12
* ISO/IEC18004:2000
* Wikipedia-Seite zu QR Codes
<http://de.wikipedia.org/wiki/QR-Code>, Stand 22.12.12
* Aufbau QR Codes und BCH Codes: <http://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders>, Stand 1.1.13
* Understanding QR Codes
<http://marksprague.wordpress.com/qr-codes-technology/understanding-qr-codes/>, Stand 3.1.13
* Abenteuer Informatik, IT zum Anfassen – von Routenplaner bis Onlinebanking, Jens Gallenbacher, 3. Auflage, 2012
* CS Unplugged
<http://www.csunplugged.org/error-detection>, Stand 3.1.13
* QR Code Generator
<http://goqr.me/de/>, Stand 1.1.13
* Bilder Design QR Codes
<http://www.qrcode-generator.de/design>, Stand 9.1.13
* ISO-8559-1:

<http://de.wikipedia.org/wiki/ISO_8859-1>, Stand 10.1.13

* Umrechnung Hex-Bin:

<http://www.table-ascii.com/>, Stand 10.1.13

* QR Code Tutorial:

<http://www.thonky.com/qr-code-tutorial/introduction/>, Stand 10.1.13

* QR Art Codes:

<http://research.swtch.com/qart>, Stand 10.3.13

* <http://qrbcn.com/imatgesbloc/Three_QR_Code.pdf>, Stand 10.3.13