Codetastic

v0.2.0 MIT

Generate all sorts of codes in Typst.

J. Neugebauer

https://github.com/jneug/typst-codetastic

codetastic draws different kinds of codes in your Typst documents. Supported codes include EAN-13 barcodes and QR-Codes.

The codes are created and drawn in pure Typst.

Part I.
Codes

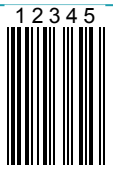
#ean13-encode(i, number, odd: none)

Encode a digit into seven bits according to the EAN-13 standard. Each digit is encoded into seven bits via one of three encoding tables, determined by `i` and `odd`.

#ean5(code, scale: 1, colors: "(white, black)")

Create an EAN-5 barcode.

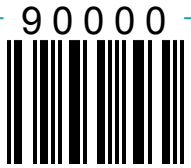
```
#codetastic.ean5(12345)
```



The `code` can be given as a five digit number in integer or string format, or as an array with five integer digits.

The size of the barcode can be scaled down to 80% and up to 200%. The height of the bars can be trimmed down to 50%.

```
#codetastic.ean5(
  scale:(1.8, .5), "90000")
```



EAN-5 codes are usually added to ean-13 codes as add-ons:

```
#grid(columns:2, row-gutter: 2pt,
  align(center, text(6pt, font:"Arial", "ISBN: 978-1-123-12345-6")), [],
  codetastic.ean13("9781123123456"), codetastic.ean5(scale:.9, "90000")
)
```

See https://www.softmatic.com/barcode-ean-13.html#ean-add-on for more information about EAN-5 codes.

Argument

`code` — `integer` | `string` | `array`

A five digit number as integer or string, or an array with five integers.

Argument

`scale: 1` `float`

Scale of the code between `0.8` and `2`.

Argument

`colors: "(white, black)"` `array`

An array with exactly two colors: background and foreground.

```
#ean8(code, scale: 1, colors: "(white, black)", lmi: "false")
```

Create an EAN-8 barcode.

```
#codetastic.ean8(2903370)
```

The `code` can be given as a seven or eight digit number in integer or string format, or as an array with seven or eight integer digits. Codes with seven digits will have the checksum value appended to the code, while for eight digit codes the given checksum is validated.

The size of the barcode can be scaled down to 80% and up to 200%. The height of the bars can be trimmed down to 50%.

```
#codetastic.ean8(
  scale:(1.8, .5), "29033706")
```

See https://www.softmatic.com/barcode-ean-8.html for more information about EAN-8 codes.

Argument

`code` `integer` | `string` | `array`

Either a seven or eight digit number as integer or string, or an array with seven or eight integers.

Argument

`scale: 1` `float`

Scale of the code between `0.8` and `2`.

Argument

`colors: "(white, black)"` `array`

An array with exactly two colors: background and foreground.

Argument

`lmi: "false"` `boolean`

If `true`, *light margin indicators* will be shown.

```
#codetastic.ean8(lmi:true,
29033706)
```

#ean13(code, scale: 1, colors: "(white, black)", lmi: "false")

Create an EAN-13 barcode.

```
#codetastic.ean13(240701400194)
```

The `code` can be given as a 12 or 13 digit number in integer or string format, or as an array with 12 or 13 integer digits. Codes with 12 digits will have the checksum value appended to the code, while for 13 digit codes the given checksum is validated.

The size of the barcode can be scaled down to 80% and up to 200%. The height of the bars can be trimmed down to 50%.

```
#codetastic.ean13(
  scale:(1.8, .5), "2407014001944")
```

See https://www.softmatic.com/barcode-ean-13.html for more information about EAN-13 codes.

Argument

`code` `integer` | `string` | `array`

Either a 12 or eight 13 number as integer or string, or an array with 12 or 13 integers.

Argument

`scale: 1` `float`

Scale of the code between `0.8` and `2`.

Argument

`colors: "(white, black)"` `array`

An array with exactly two colors: background and foreground.

Argument

`lmi: "false"` `boolean`

If `true`, a *light margin indicator* will be shown.

```
#codetastic.ean13(lmi:true,
9781234567897)
```

#upc-a(code, scale: 1, colors: "(white, black)", lmi: "false")

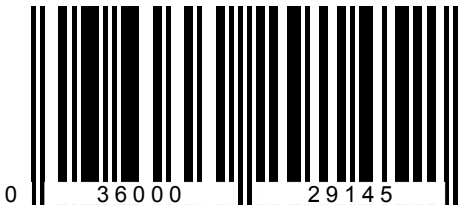
Create an UPC-A barcode.

```
#codetastic.upc-a("03600029145")
```

The `code` can be given as a 11 or 12 digit number in integer or string format, or as an array with 11 or 12 integer digits. Codes with 11 digits will have the checksum value appended to the code, while for 12 digit codes the given checksum is validated.

The size of the barcode can be scaled down to 80% and up to 200%. The height of the bars can be trimmed down to 50%.

```
#codetastic.upc-a(
  scale:(1.8, .5), "03600029145")
```



See https://www.softmatic.com/barcode-upc-a.html for more information about UPC-A codes.

Argument

`code` `integer` | `string` | `array`

Either a 11 or eight 12 number as integer or string, or an array with 11 or 12 integers.

Argument

`scale: 1` `float`

Scale of the code between `0.8` and `2`.

Argument

`colors: "(white, black)"` `array`

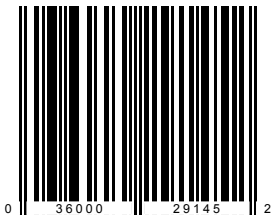
An array with exactly two colors: background and foreground.

Argument

`lmi: "false"` `boolean`

If `true`, a *light margin indicator* will be shown.

```
#codetastic.upc-a(lmi:true,
"03600029145")
```




```
#qrcode(
  data,
  quiet-zone: 4,
  min-version: 1,
  ecl: ""l"",
  mask: auto,
  size: auto,
  width: auto,
  colors: "(white, black)",
  debug: "false"
)
```

Draws a QR-Code encoding `data`.

```
#codetastic.qrcode("https://qrcode.com/en")
```

Some caveats:

- The generation of larger codes can take quit some time.
- To speed-up compilation times, calculations for the optimal masking patterns don't the same approach as defined in the qr-code documentation. Codes will still be valid, but might look differnt than the output of other generators.
- Kanji and ECI encodings are not yet supported. Maybe they will be in the future.
- UTF-8 is not supported.

Improving speed for larger code versions:
Generating qr-codes with large amount of data can take long. Calculating the best masking pattern to produce the most readable code is currently one of the most expensive calculations in code creation. This can be mitigated by manually setting the mask to use. To do so, follow these steps:

- Compile your document while passing `debug: true` to `#qrcode()`.
- After compilation finished, look for the code in your output and note the mask number shown above the code.
- Remove the `debug` argument and set `mask` to the mask number.

Now the code creation will skip detection of the optimal mask and use the passed in mask. This should speed-up compilation times considerably.

Argument

`data` `string`

The data to encode.

Argument

`quiet-zone: 4` `integer`

Whitespace around the code in number of modules. The qr-code standard suggests a quiet zone of at least four modules.

Argument

`min-version: 1` `integer`

Minimum version for the code. A number between 1 and 40. If `data` is to large for the minimum code verison, the next larger verison that fits the data is selected.

Argument

`ecl: ""l""` `string`

Error correction level. One of `"l"`, `"m"`, `"q"` or `"h"`.

Argument

`mask: auto` `auto` | `integer`

Forces a mask to apply to the code. Number between (0 and 7). For `auto` the best mask is selected according to the qr-code standard.

Argument

`size: auto` `auto` | `length`

Size of a modules square.

Argument

`width: auto` `auto` | `length`

If set to a length, the module size will be adjusted to create a qr-code with the given width. Will overwrite any setting for `size`.

Argument

```
colors: "(white, black)"                                    array
```

An array with exactly two colors: background and foreground.