# The 9th Bit: Encodings in Ruby 1.9

Norman Clarke

## Encoding API

One of the most visible changes to Ruby in 1.9

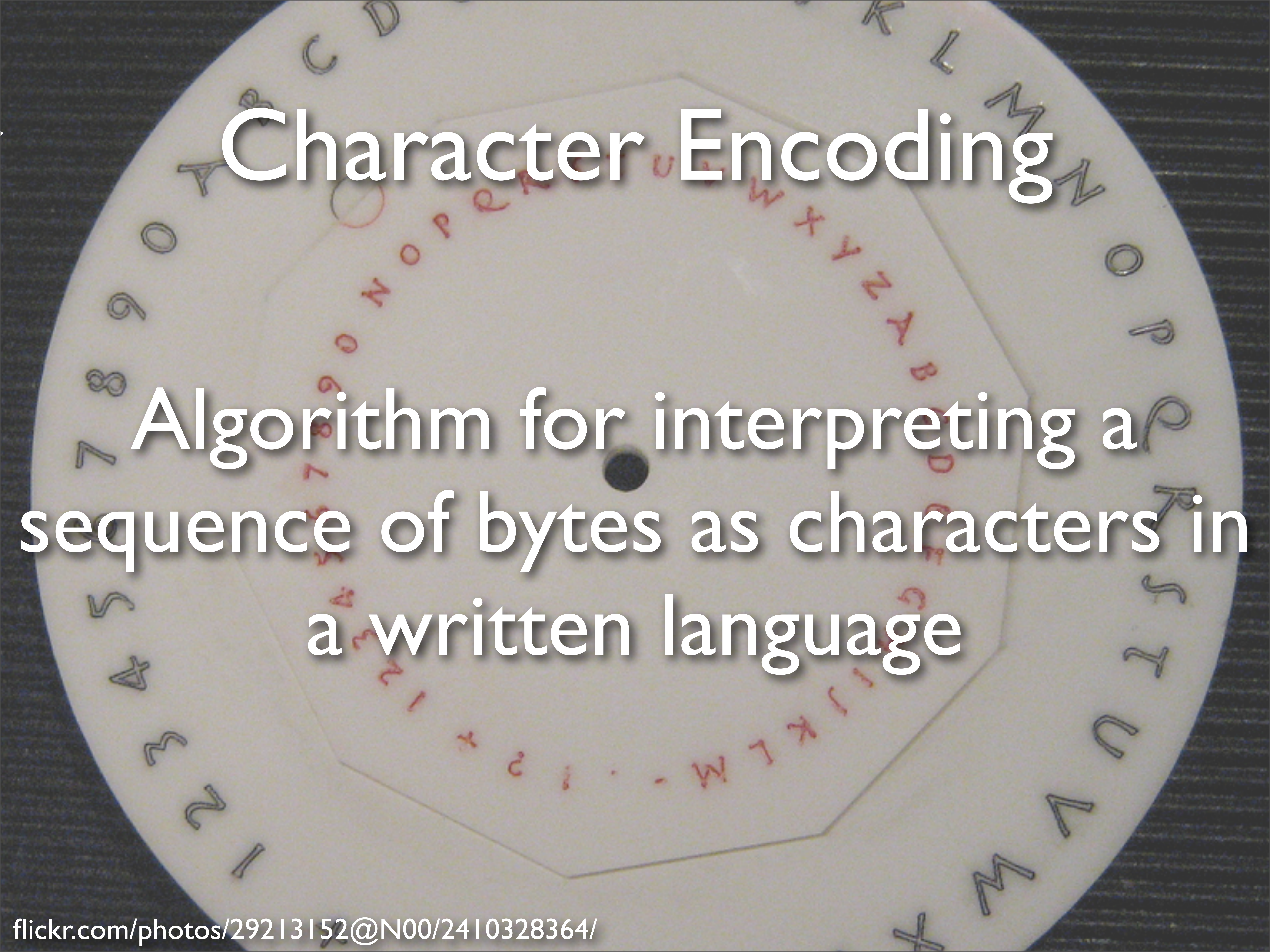
```
invalid byte sequence
in US-ASCII/UTF8

invalid multibyte char
(US-ASCII)

`encode':
"\xE2\x80\xA6" from
UTF-8 to ISO-8859-1
```

## Today’s Topics

- Character Encodings
- Ruby’s Encoding API
- Avoiding problems with UTF-8



## Character Encoding

Algorithm for interpreting a sequence of bytes as characters in a written language

flickr.com/photos/29213152@N00/2410328364/

# ASCII

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | 1 | soh | 2 | stx | 3 | etx | 4 | eot | 5 | enq | 6 | ack | 7 | bel |
| 8 | bs | 9 | ht | 10 | nl | 11 | vt | 12 | np | 13 | cr | 14 | so | 15 | si |
| 16 | dle | 17 | dc1 | 18 | dc2 | 19 | dc3 | 20 | dc4 | 21 | nak | 22 | syn | 23 | etb |
| 24 | can | 25 | em | 26 | sub | 27 | esc | 28 | fs | 29 | gs | 30 | rs | 31 | us |
| 32 | sp | 33 | ! | 34 | " | 35 | # | 36 | $ | 37 | % | 38 | & | 39 | ' |
| 40 | ( | 41 | ) | 42 | * | 43 | + | 44 | , | 45 | - | 46 | . | 47 | / |
| 48 | 0 | 49 | 1 | 50 | 2 | 51 | 3 | 52 | 4 | 53 | 5 | 54 | 6 | 55 | 7 |
| 56 | 8 | 57 | 9 | 58 | : | 59 | ; | 60 | < | 61 | = | 62 | > | 63 | ? |
| 64 | @ | 65 | A | 66 | B | 67 | C | 68 | D | 69 | E | 70 | F | 71 | G |
| 72 | H | 73 | I | 74 | J | 75 | K | 76 | L | 77 | M | 78 | N | 79 | O |
| 80 | P | 81 | Q | 82 | R | 83 | S | 84 | T | 85 | U | 86 | V | 87 | W |
| 88 | X | 89 | Y | 90 | Z | 91 | [ | 92 | \ | 93 | ] | 94 | ^ | 95 | _ |
| 96 | ` | 97 | a | 98 | b | 99 | c | 100 | d | 101 | e | 102 | f | 103 | g |
| 104 | h | 105 | i | 106 | j | 107 | k | 108 | l | 109 | m | 110 | n | 111 | o |
| 112 | p | 113 | q | 114 | r | 115 | s | 116 | t | 117 | u | 118 | v | 119 | w |
| 120 | x | 121 | y | 122 | z | 123 | { | 124 | \| | 125 | } | 126 | ~ | 127 | del |

## ASCII: 7 bits

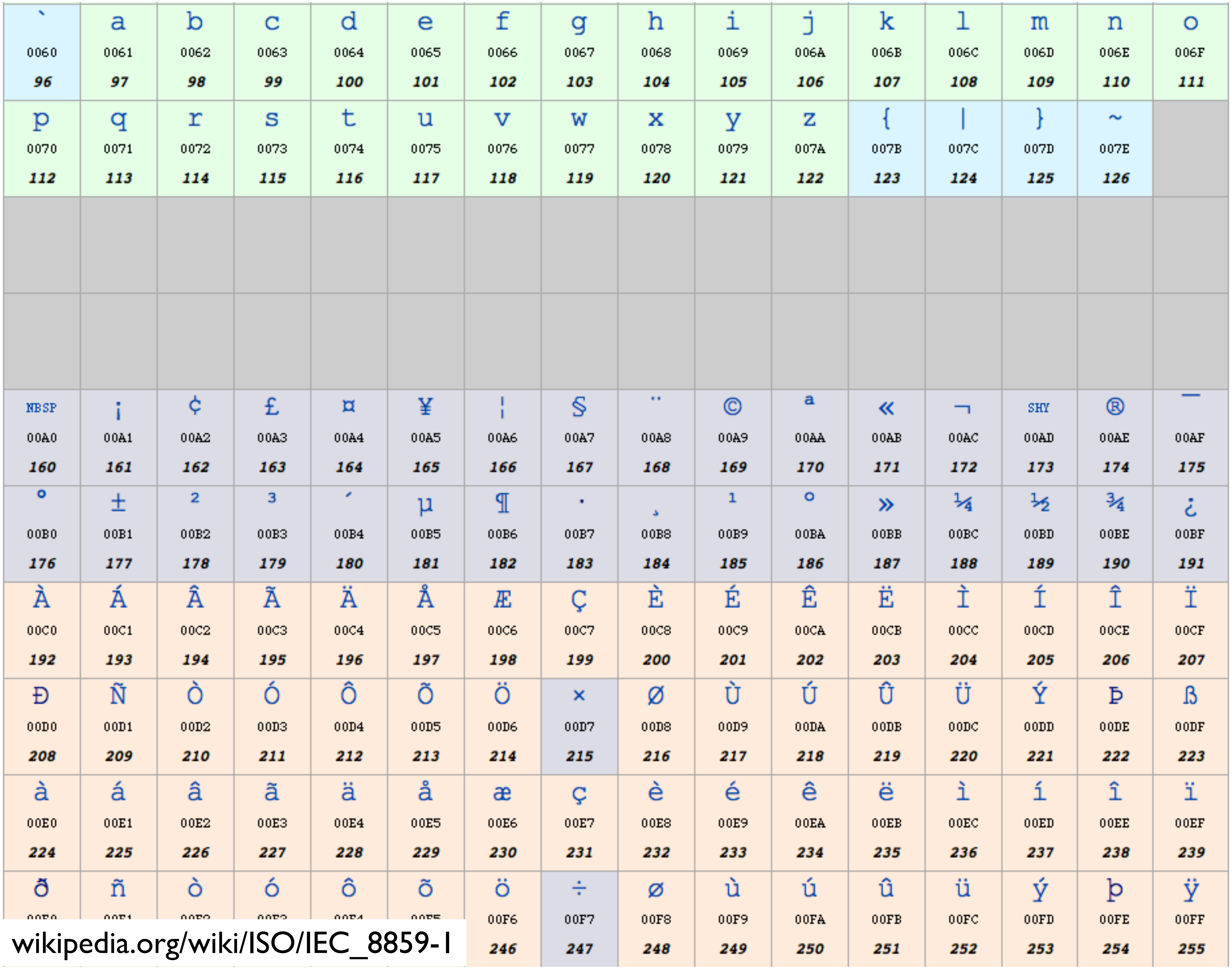
a

```
97: 0110 0001
```

## Latin1: 8 bits

ã

`227: 1110 0011`



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ` 0060 96 | a 0061 97 | b 0062 98 | c 0063 99 | d 0064 100 | e 0065 101 | f 0066 102 | g 0067 103 | h 0068 104 | i 0069 105 | j 006A 106 | k 006B 107 | l 006C 108 | m 006D 109 | n 006E 110 | o 006F 111 |
| p 0070 112 | q 0071 113 | r 0072 114 | s 0073 115 | t 0074 116 | u 0075 117 | v 0076 118 | w 0077 119 | x 0078 120 | y 0079 121 | z 007A 122 | { 007B 123 | \| 007C 124 | } 007D 125 | ~ 007E 126 | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| NBSP 00A0 160 | ¡ 00A1 161 | ¢ 00A2 162 | £ 00A3 163 | ¤ 00A4 164 | ¥ 00A5 165 | ¦ 00A6 166 | § 00A7 167 | ¨ 00A8 168 | © 00A9 169 | ª 00AA 170 | « 00AB 171 | ¬ 00AC 172 | SHY 00AD 173 | ® 00AE 174 | ¯ 00AF 175 |
| ° 00B0 176 | ± 00B1 177 | ² 00B2 178 | ³ 00B3 179 | ´ 00B4 180 | µ 00B5 181 | ¶ 00B6 182 | · 00B7 183 | ¸ 00B8 184 | ¹ 00B9 185 | º 00BA 186 | » 00BB 187 | ¼ 00BC 188 | ½ 00BD 189 | ¾ 00BE 190 | ¿ 00BF 191 |
| À 00C0 192 | Á 00C1 193 | Â 00C2 194 | Ã 00C3 195 | Ä 00C4 196 | Å 00C5 197 | Æ 00C6 198 | Ç 00C7 199 | È 00C8 200 | É 00C9 201 | Ê 00CA 202 | Ë 00CB 203 | Ì 00CC 204 | Í 00CD 205 | Î 00CE 206 | Ï 00CF 207 |
| Ð 00D0 208 | Ñ 00D1 209 | Ò 00D2 210 | Ó 00D3 211 | Ô 00D4 212 | Õ 00D5 213 | Ö 00D6 214 | × 00D7 215 | Ø 00D8 216 | Ù 00D9 217 | Ú 00DA 218 | Û 00DB 219 | Ü 00DC 220 | Ý 00DD 221 | Þ 00DE 222 | ß 00DF 223 |
| à 00E0 224 | á 00E1 225 | â 00E2 226 | ã 00E3 227 | ä 00E4 228 | å 00E5 229 | æ 00E6 230 | ç 00E7 231 | è 00E8 232 | é 00E9 233 | ê 00EA 234 | ë 00EB 235 | ì 00EC 236 | í 00ED 237 | î 00EE 238 | ï 00EF 239 |
| ð | ñ | ò | ó | ô | õ | ö 00F6 246 | ÷ 00F7 247 | ø 00F8 248 | ù 00F9 249 | ú 00FA 250 | û 00FB 251 | ü 00FC 252 | ý 00FD 253 | þ 00FE 254 | ÿ 00FF 255 |

wikipedia.org/wiki/ISO/IEC_8859-1

## Other 8-bit Encodings

Work for most languages

中文

字喃/𡦂喃/𡨸喃

# 256 is not enough for:

- Chinese
- Japanese
- some others

かたかな

한국어/조선말

fanpop.com/spots/gandalf/images/7018563/title/gandalf-vs-el-balrog-fanart

# 8-bit overlap

202

Ê Ę Ъ ت К ส Ź

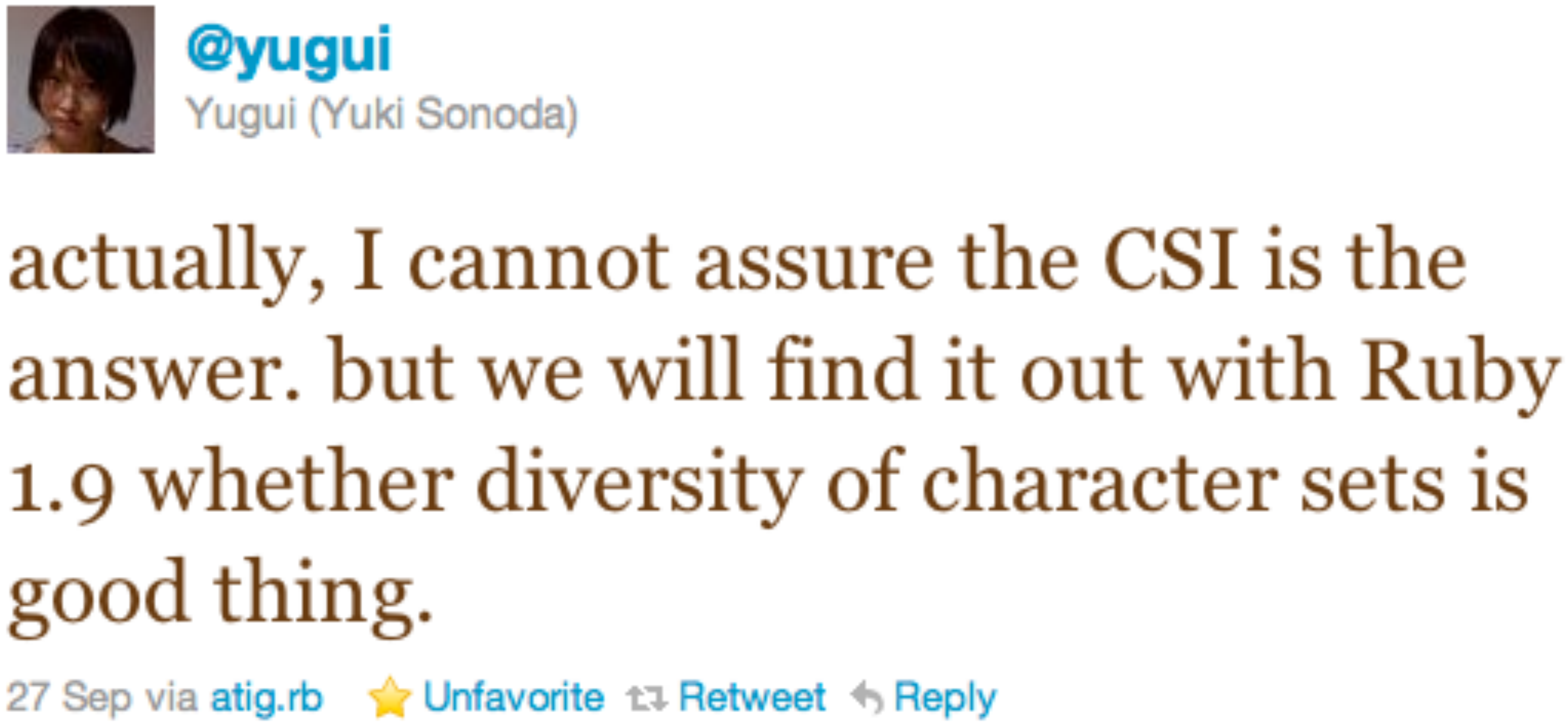
## Unicode: An Improbable Success

`¡cn:中文!`

Used internally by Perl, Java, Python 3, Haskell and others

## Unicode in Japan: not as popular

## Ruby 1.9: Character Set Independence



@yugui
Yugui (Yuki Sonoda)

actually, I cannot assure the CSI is the answer. but we will find it out with Ruby 1.9 whether diversity of character sets is good thing.

27 Sep via atig.rb Unfavorite Retweet Reply

## Ruby’s Encoding API

- Source code
- String
- Regexp
- IO
- Encoding

# Source

```ruby
# coding: utf-8
class Canção
  GÊNEROS = [:forró, :carimbó, :afoxé]
  attr_accessor :gênero
end
asa_branca = Canção.new
asa_branca.gênero = :forró
p asa_branca.gênero
```

## Warnings

- Breaks syntax highlighting
- #inspect, #p don't work as of 1.9.2
- Some editors/programmers will probably mess up your code
- Just because you can, doesn't mean you should

## String

```
# encoding: utf-8
string = “ã”
string.length        #=> 1
string.bytesize      #=> 2
string.bytes.to_a    #=> [195, 163]
string.encode! "ISO-8859-1"
string.length        #=> 1
string.bytesize      #=> 1
string.bytes.to_a    #=> [227]
```

# String

```
# encoding: utf-8
string = “ã”
string.length         #=> 1
string.bytesize       #=> 2
string.bytes.to_a     #=> [195, 163]
string.encode! "ISO-8859-1"
string.length         #=> 1
string.bytesize       #=> 1
string.bytes.to_a     #=> [227]
```

## String

```
# encoding: utf-8
string = “ã”
string.length         #=> 1
string.bytesize       #=> 2
string.bytes.to_a     #=> [195, 163]
string.encode! "ISO-8859-1"
string.length         #=> 1
string.bytesize       #=> 1
string.bytes.to_a     #=> [227]
```

# String

```
# encoding: utf-8
string = “ã”
string.length        #=> 1
string.bytesize      #=> 2
string.bytes.to_a    #=> [195, 163]
string.encode! "ISO-8859-1"
string.length        #=> 1
string.bytesize      #=> 1
string.bytes.to_a    #=> [227]
```

## String

```
puts a1 ("ã")
puts a2 ("ã")
a1.encoding                #=> "ASCII-8BIT"
a2.encoding                #=> "UTF-8"
a1.bytes.to_a == a2.bytes.to_a #=> true
a1 == a2                   #=> false
```

# Regexp

```
# vim: set fileencoding=utf-8

pat = /ã/
pat.encoding                 #=> “UTF-8”
pat.encode! “ISO-8859-1” #=> FAIL
pat = “ã”.encode “ISO-8859-1”
regexp = Regexp.new(pat) #=> OK
```

## Regexp

```
# vim: set fileencoding=utf-8

pat = /ã/
pat.encoding                     #=> “UTF-8”
pat.encode! “ISO-8859-1” #=> FAIL
pat = “ã”.encode “ISO-8859-1”
regexp = Regexp.new(pat) #=> OK
```

## Regexp

```
# vim: set fileencoding=utf-8

pat = /ã/
pat.encoding                     #=> “UTF-8”
pat.encode! “ISO-8859-1” #=> FAIL
pat = “ã”.encode “ISO-8859-1”
regexp = Regexp.new(pat) #=> OK
```

## IO

```
f = File.open("file.txt", "r:ISO-8859-1")
data = f.read
data.encoding #=> “ ISO-8859-1”
```

# IO

```
f = File.open("file.txt", "rb:UTF-16BE:UTF8")
data = f.read
data.encoding #=> “UTF-8”
```

## IO

```
f = File.open("file.txt", "r:BINARY") # (or “rb”)
data = f.read
data.encoding #=> “ASCII-8BIT”
data.force_encoding "UTF-8"
```

# IO

```
f = File.open("file.txt", "r:BINARY") # (or “rb”)
data = f.read
data.encoding #=> “ASCII-8BIT”
data.force_encoding "UTF-8"
```

## IO

```
f = File.open("file.txt", "r:BINARY") # (or “rb”)
data = f.read
data.encoding #=> “ASCII-8BIT”
data.force_encoding "UTF-8"
```

# Encoding

```
Encoding.list.size #=> 95
Encoding.default_external = "ISO-8859-1"
Encoding.default_internal = "UTF-8"

File.open("latin1.txt", "r") do |file|
  p file.external_encoding #=> ISO-8859-1
  data = file.read
  p data.encoding          #=> UTF-8
end
```

# Encoding

```
Encoding.list.size #=> 95
Encoding.default_external = "ISO-8859-1"
Encoding.default_internal = "UTF-8"

File.open("latin1.txt", "r") do |file|
  p file.external_encoding #=> ISO-8859-1
  data = file.read
  p data.encoding          #=> UTF-8
end
```

# Encoding

```
Encoding.list.size #=> 95
Encoding.default_external = "ISO-8859-1"
Encoding.default_internal = "UTF-8"

File.open("latin1.txt", "r") do |file|
  p file.external_encoding #=> ISO-8859-1
  data = file.read
  p data.encoding          #=> UTF-8
end
```

# Encoding

```
Encoding.list.size #=> 95
Encoding.default_external = "ISO-8859-1"
Encoding.default_internal = "UTF-8"

File.open("latin1.txt", "r") do |file|
  p file.external_encoding #=> ISO-8859-1
  data = file.read
  p data.encoding          #=> UTF-8
end
```

## UTF-8: a Unicode Encoding

Unicode, UTF-8, UTF-16, UTF-32, UCS-2, etc.

## UTF-8

Backwards-compatible with ASCII

## Make UTF-8 your default option

## UTF-8 and HTML

```
<meta http-
equiv="content-type"
content="text/
html;charset=UTF-8" />
```

## UTF-8 and HTML

日本語

## UTF-8 and HTML

æ—¥æœ¬èªž

## UTF-8 and HTML

```
<form action="/"
accept-
charset="UTF-8">
```

## UTF-8 and HTML

```
f.html?l=日本語
```

# UTF-8 and HTML

```
f.html?l=
%26%23260085%3B
%26%23264412%3B
%26%2335
```

...here's where things get kind of strange.

# Case Folding

```
“JOÃO”.downcase #=> “joÃo”
“joão”.upcase   #=> “JOãO”
```

## Case Folding

```
# Unicode
Unicode.downcase(“JOÃO”)

# Active Support
“JOÃO”.mb_chars.downcase
```

## Equivalence

```
# NOT always true
"João" == "João"
```

## Two ways to represent many characters

`"ã" or "a" + "~"`

## Composed

```
a = Unicode.normalize_C("ã")
a.bytes.to_a #=> [195, 163]
```

# Decomposed

```
a = Unicode.normalize_D("ã")
a.bytes.to_a #=> [97, 204, 131]
```

# Why?

```
dec = Unicode.normalize_D("ã")
dec =~ /a/  # match

comp = Unicode.normalize_C("ã")
comp =~ /a/ # no match
```

## Normalize string keys!!!

## You have been warned

```
{
 "João" => "authorized",
 "João" => "not authorized"
}
```

## Some libraries

- Unicode
- Active Support
- Java's stdlib

## Cleaning up bad data: avoid Iconv

## Tidy Bytes

```
require "active_support"
require "active_support/multibyte/unicode"

include ActiveSupport::Multibyte
Unicode.tidy_bytes(@bad_string)
```

## MySQL

Set encoding options early

## Approximating ASCII: "João" => "joao"

```
# 1: decompose
@s = Unicode.normalize_D(@s)

# 2: delete accent marks
@s.gsub!(/[^\x00-\x7F]/, '')

# 3: FAIL
```

## OK

ã á ê ü à ç

a a e u a c

## FAIL

ß ø œ æ

"" "" "" ""

## Use instead:

- Active Support's Inflector.transliterate
- I18n.transliterate
- Babosa

## To Sum Up...

Ruby is weird

## Use UTF-8

## Normalize UTF-8 keys

## Configure MySQL properly for UTF-8

# THANKS!

github.com/norman/enc

@compay

norman@njclarke.com