# Unicode + PHP

Ayesh Karunaratne

# Unicode + PHP


# Ayesh Karunaratne

Security Researcher, Freelance Software Developer

Kandy, Sri Lanka - Everywhere

https://ayesh.me

Ayesh

@Ayeshlive

Ayesh

https://ayesh.me/talk/Unicode

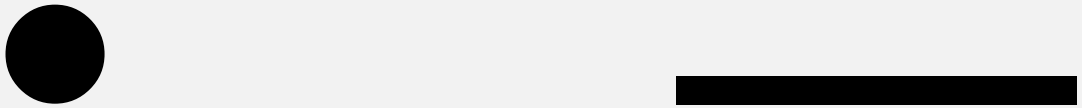
# Morse Code

# Morse Code

***Represent*** English characters, numbers, with short and long pulses

# Morse Code

***Represent*** English characters, numbers, with short and long pulses



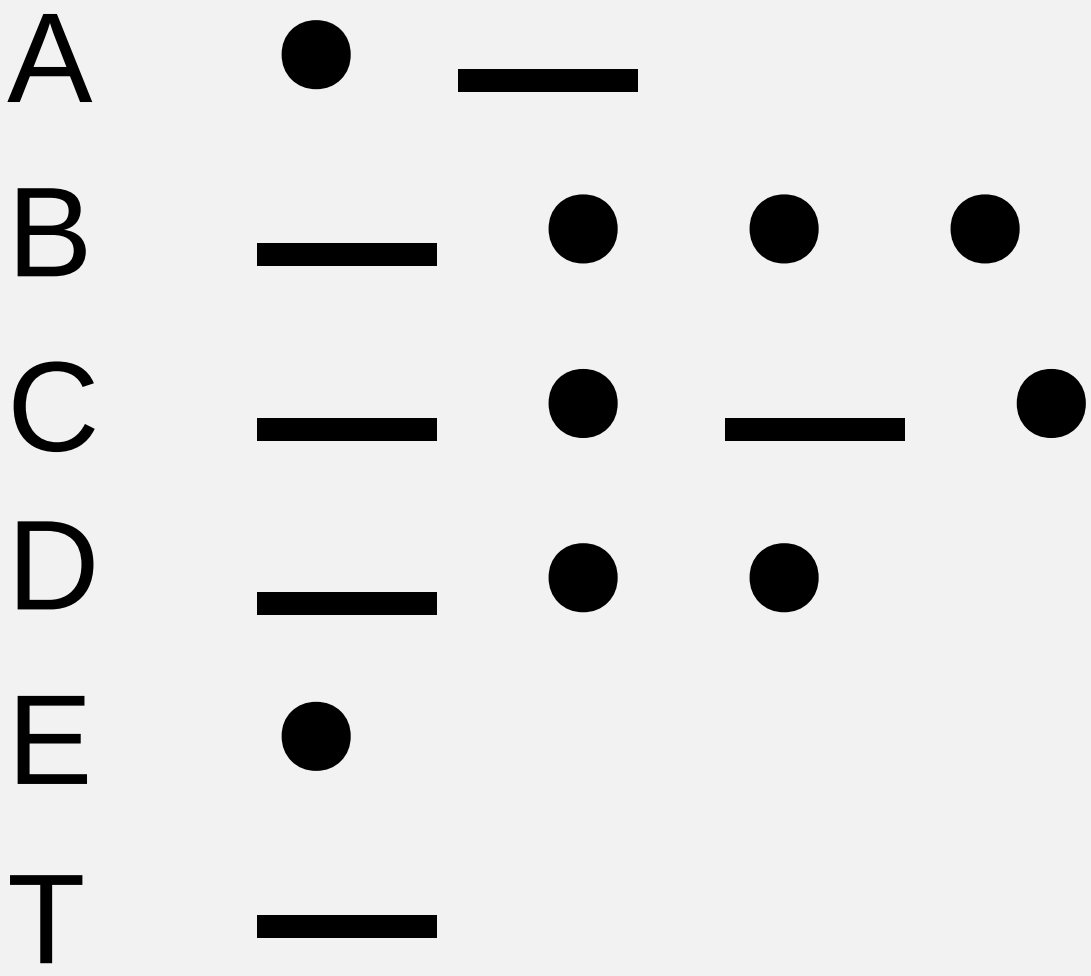
Short Beep

Long Beep

Silence

# Morse Code

***Represent*** English characters, numbers, with short and long pulses




·
Short Beep

—
Long Beep

Silence

# Morse Code

***Represent*** English characters, numbers, with short and long pulses

CAB

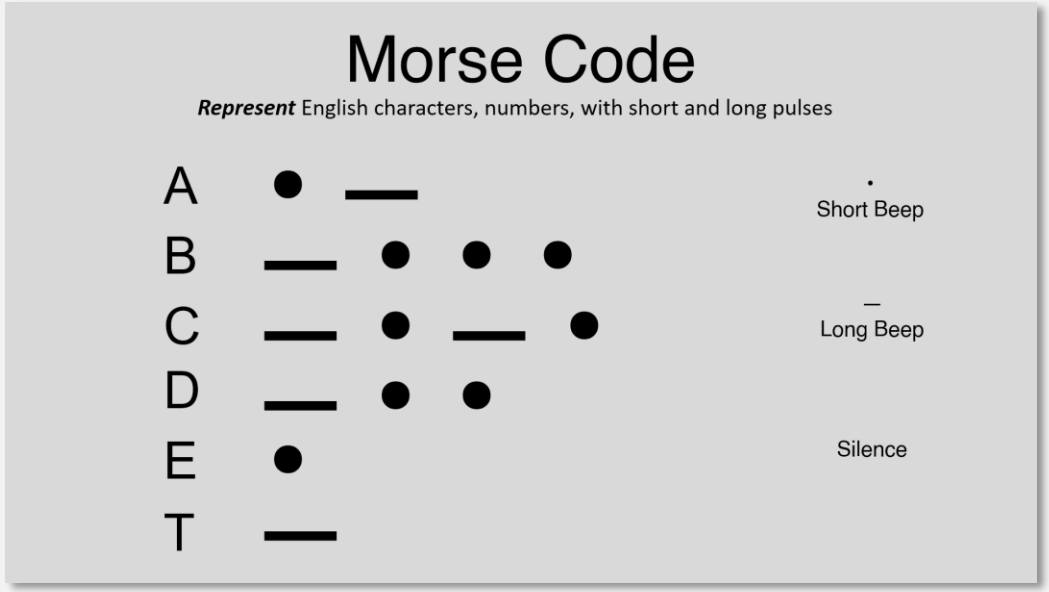
TED

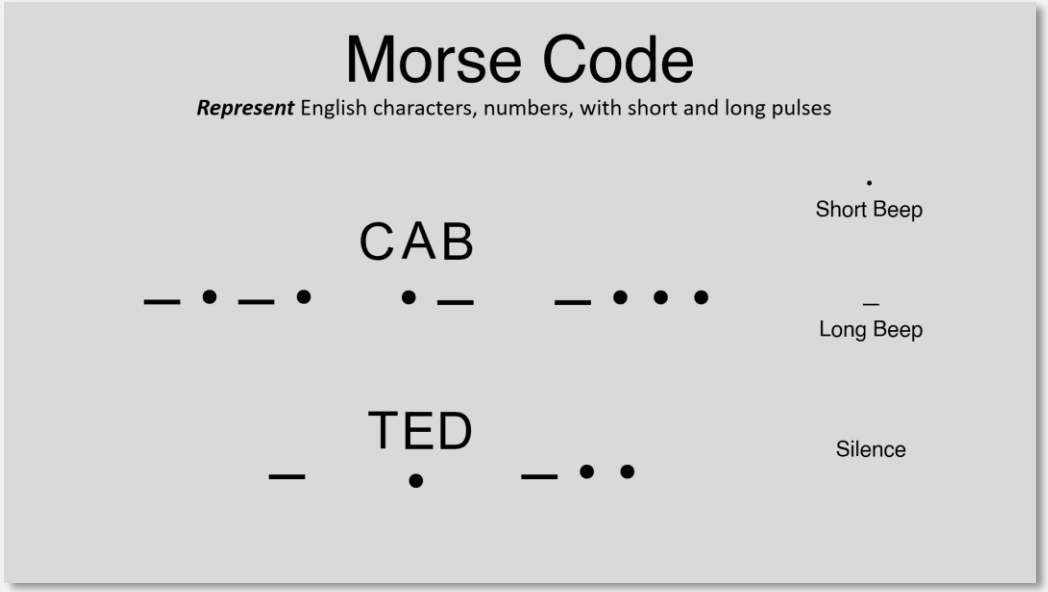
Short Beep

Long Beep

Silence

# Character Encoding



# How do you Encode Characters in a Computer System?

# How do you Encode Characters in a Computer System?

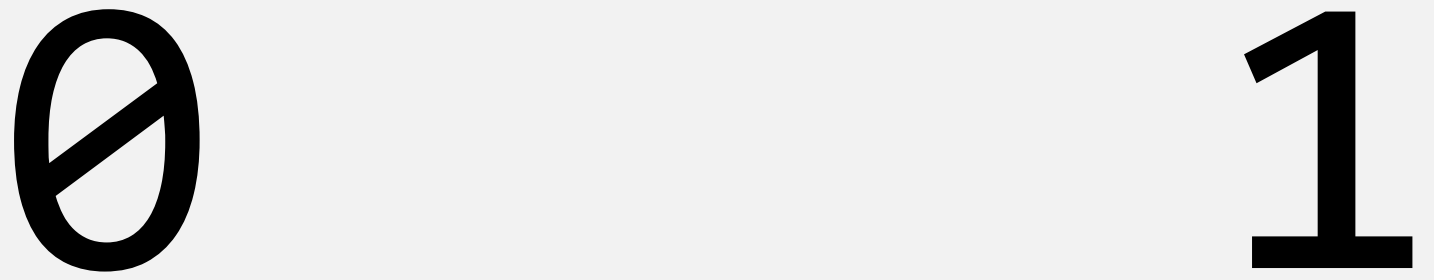
0 1


The First Official Photograph of the United States Senate in Session

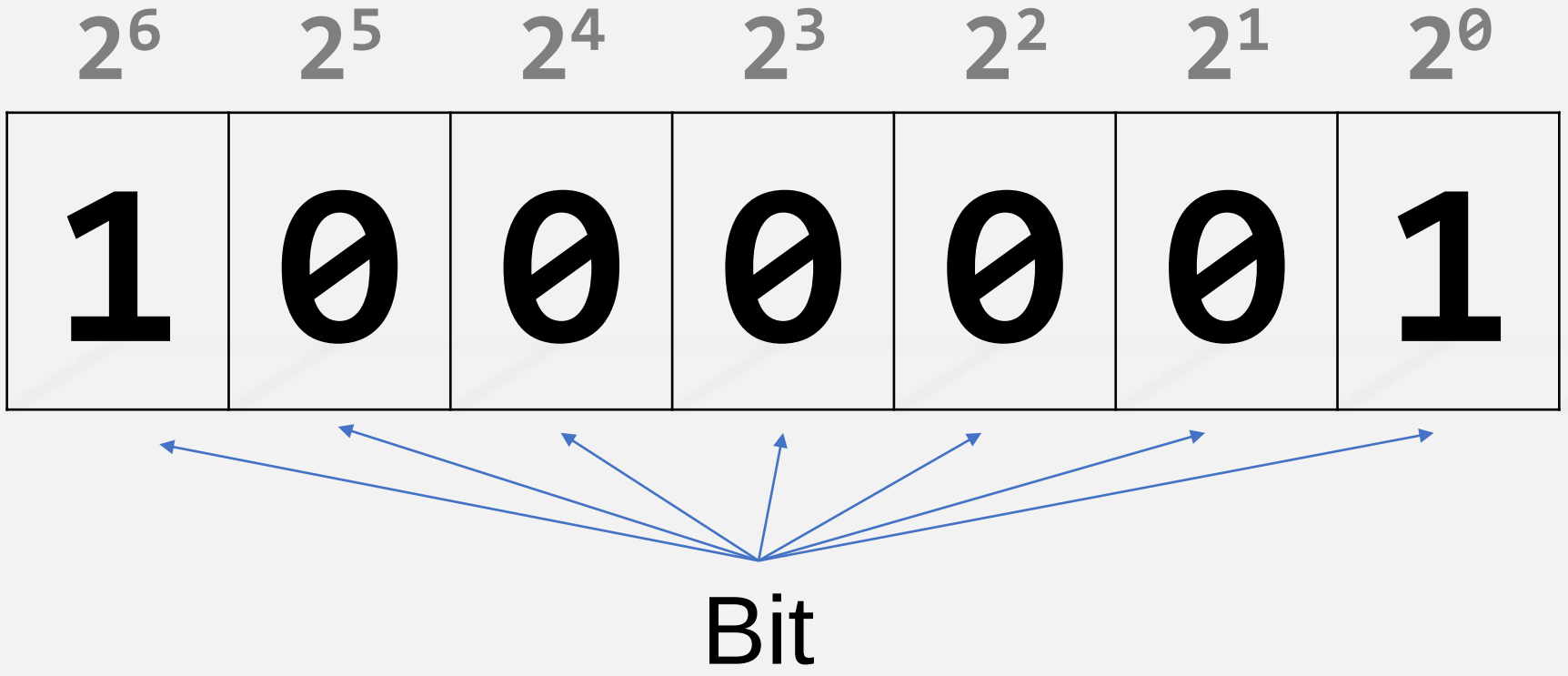
https://uschs.wordpress.com/2013/09/25/september-24-1963-the-first-official-photograph-of-the-united-states-senate-in-session

# ASCII

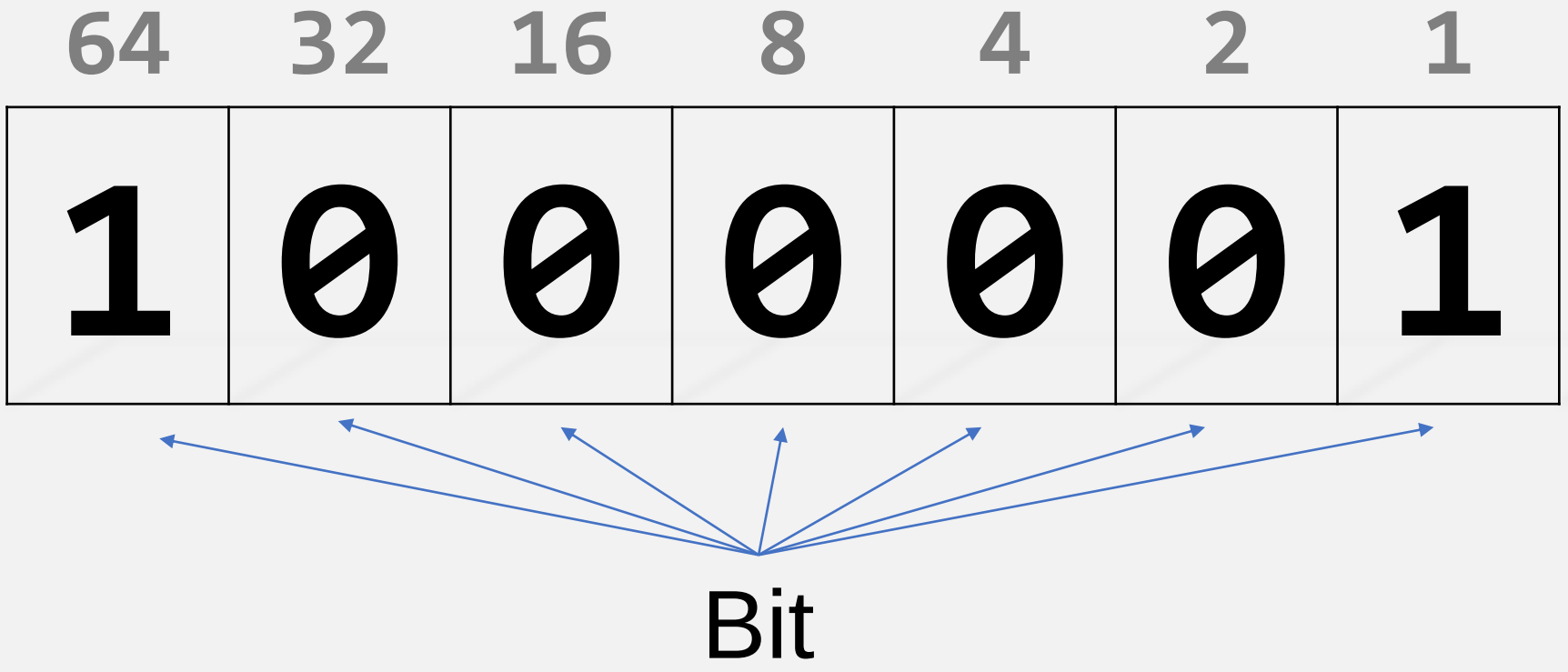
American Standard Code for Information Interchange


0
1

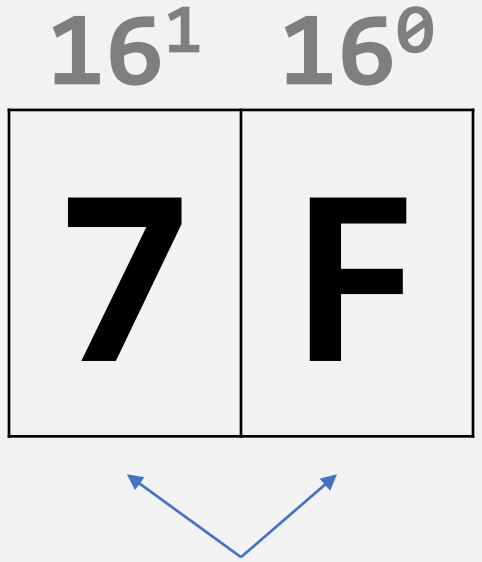
# Binary



# Binary



# Hexadecimal



# ASCII Standard

null 0

Hexadecimal

| 16 | 1 |
|---|---|
| 0 | 0 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# ASCII Standard

A

65

Hexadecimal

| 16 | 1 |
|---|---|
| 4 | 1 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |

# ASCII Standard

B

66

Hexadecimal

| 16 | 1 |
|---|---|
| 4 | 2 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |

# ASCII Standard

C

67

Hexadecimal

| 16 | 1 |
|---|---|
| 4 | 3 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |

# ASCII Standard

D

68

Hexadecimal

| 16 | 1 |
|---|---|
| 4 | 4 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |

# ASCII Standard

a

97

Hexadecimal

| 16 | 1 |
|---|---|
| 6 | 1 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |

# ASCII Standard

b

98

Hexadecimal

| 16 | 1 |
|---|---|
| 6 | 2 |

Binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |

# ASCII Standard

C

99

Hexadecimal

| 16 | 1 |
|---|---|
| 6 | 3 |

Binary

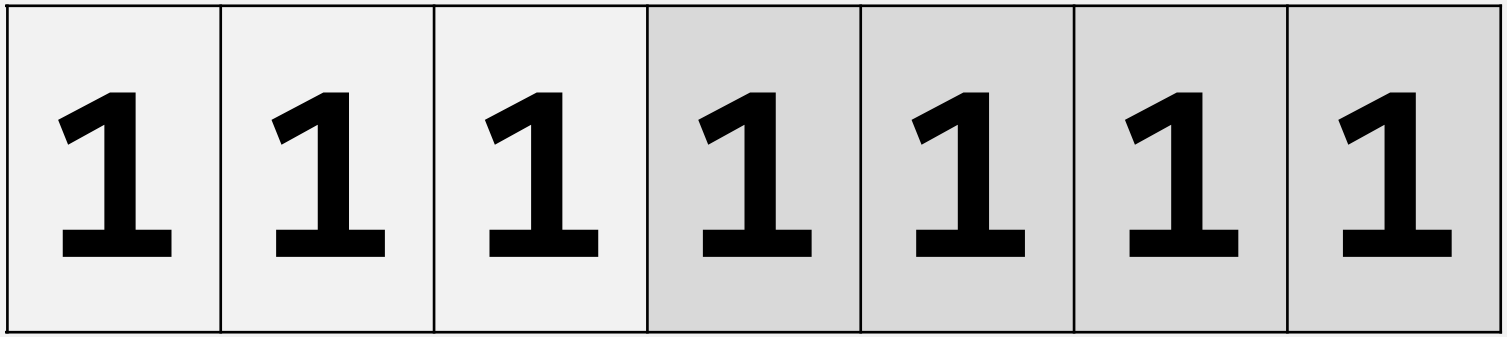
| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |

# ASCII Standard

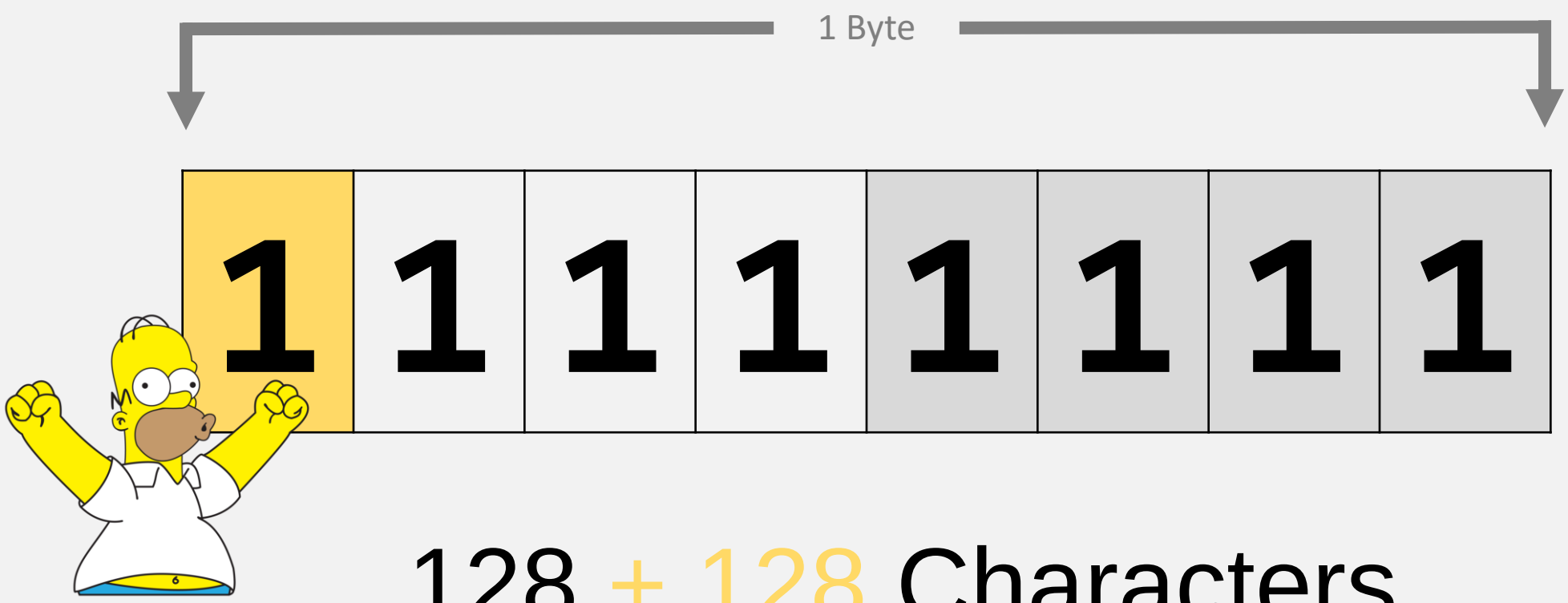
| | | Hex | Binary |
|---|---|---|---|
| A | 65 | 41 | 100 0001 |
| B | 66 | 42 | 100 0010 |
| C | 67 | 43 | 100 0011 |
| D | 68 | 44 | 100 0100 |
| E | 69 | 45 | 100 0101 |
| F | 70 | 46 | 100 0110 |
| … | … | … | … |
| a | 97 | 61 | 110 0001 |
| b | 98 | 62 | 110 0010 |
| c | 99 | 63 | 110 0011 |
| d | 100 | 64 | 110 0100 |
| e | 101 | 65 | 110 0101 |

| | | Hex | Binary |
|---|---|---|---|
| CR | 13 | D | 000 1101 |
| ESC | 27 | 1B | 001 1011 |
| *Space* | 32 | 20 | 010 0000 |
| < | 60 | 3C | 011 1100 |
| ? | 63 | 3F | 011 1111 |
| 0 | 48 | 30 | 011 0000 |
| 1 | 49 | 31 | 011 0001 |
| 2 | 50 | 32 | 011 0010 |
| 3 | 51 | 33 | 011 0011 |
| + | 43 | 2B | 010 1011 |
| / | 47 | 2F | 010 1111 |
| DEL | 127 | 7F | 111 1111 |


MUCH
LATER




128 Characters




128 + 128 Characters

MUCH, MUCH LATER...

*“Let’s include more characters”*

***“Let’s include more characters”***

- Everyone

***“Let’s include more characters”***

- Everyone, at the same time

Extended Latin-8

GB18030

JIS X 0201

Mac OS Roman

EBCDIC-500

JIS X 0208

ISO 8859-5

EBCDIC 930

IBM 949

EBCDIC-1154

ISO 8859-1

ArmSCII

EBCDIC 875

IBM AIX 367

ISO 8859-2

PETSCII

EBCDIC-1149

ISO 8859

GSM 03.38

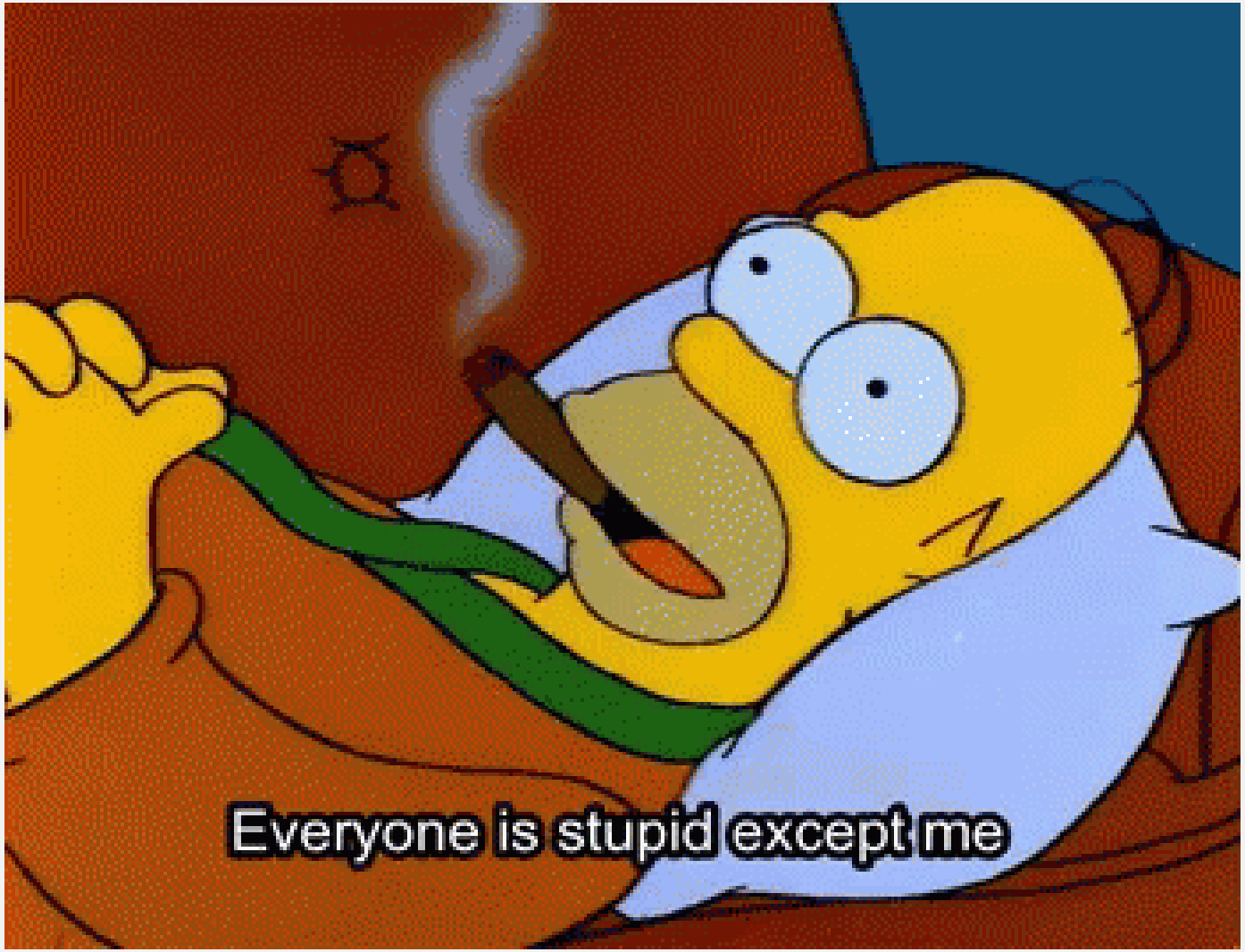
DOS МИК

CWI-2

Windows-1252

HP Roman Extension

Shit JIS


Everyone is stupid except me

# 400+ Character Encoding Standards

400+ Character Encoding Standards

Each incompatible with the rest

400+ Character Encoding Standards

Each incompatible with the rest

Only one character encoding per document

400+ Character Encoding Standards

Each incompatible with the rest

Only one character encoding per document

Mojibake

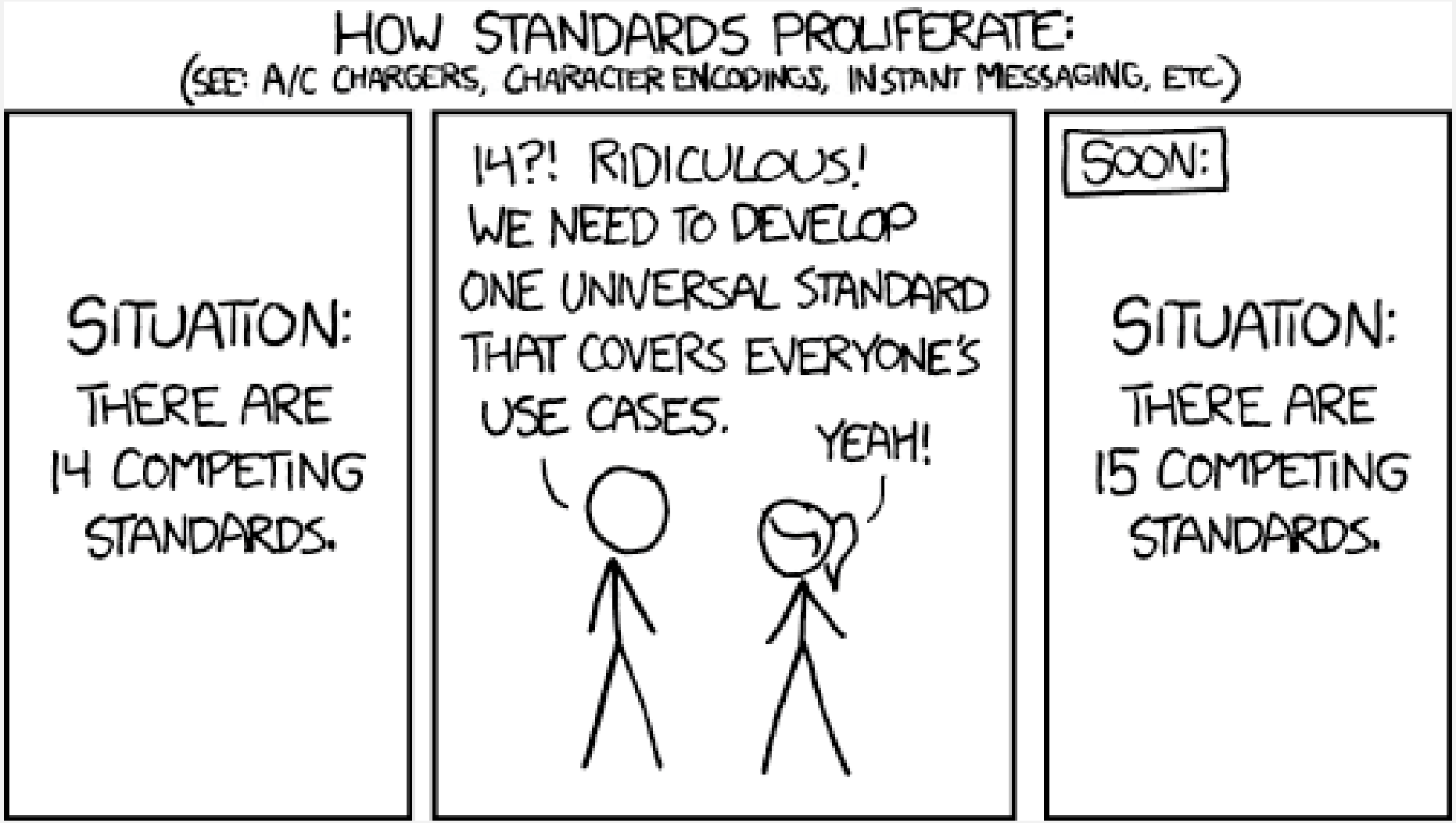
文字化け

# Mojibake

文字化け

الإعلان العالمى لحقوق الإنسان

| | |
|---|---|
| **Windows-1252** | ï»¿Ø§Ù"Ø¥Ø¹Ù"Ø§Ù† Ø§Ù"Ø¹Ø§Ù"Ù…Ù‰ Ù"ØÙ,Ùˆ, Ø§Ù"Ø¥Ù†Ø³Ø§Ù† |
| **KOI8-R** | O╩©ь╖ ы └ь ╓ь╧ы └ь╖ ы ├ ь╖ ы └ь╧ь╖ ы └ы┘ ы┴ ы └ь╜ ы ┌ыт┬ы ┌ ь╖ ы └ь ╓ы ├ьЁь╖ ы ├ |
| **ISO 8859-5** | яЛПиЇй�иSиЙй�иЇй� иЇй�иЙиЇй�й�й� й�ий�й�й�й� иЇй�иSй�иГиЇй� |
| **CP 866** | я╗┐ ╪з┘ Д╪е╪╣┘ Д╪з┘ Ж ╪з┘ Д╪╣╪з┘ Д┘ Е┘ Й ┘ Д╪н┘ В┘ И┘ В ╪з┘ Д╪е┘ Ж╪▒ ╪з┘ Ж |
| **ISO 8859-2** | ďťžŘ§Ů�ŘĽŘšŮ�Ř§Ů� Ř§Ů�ŘšŘ§Ů�Ů�Ů� Ů�ŘŮ�Ů�Ů�Ů� Ř§Ů�ŘĽŮ�ŘłŘ§Ů� |


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

# United States

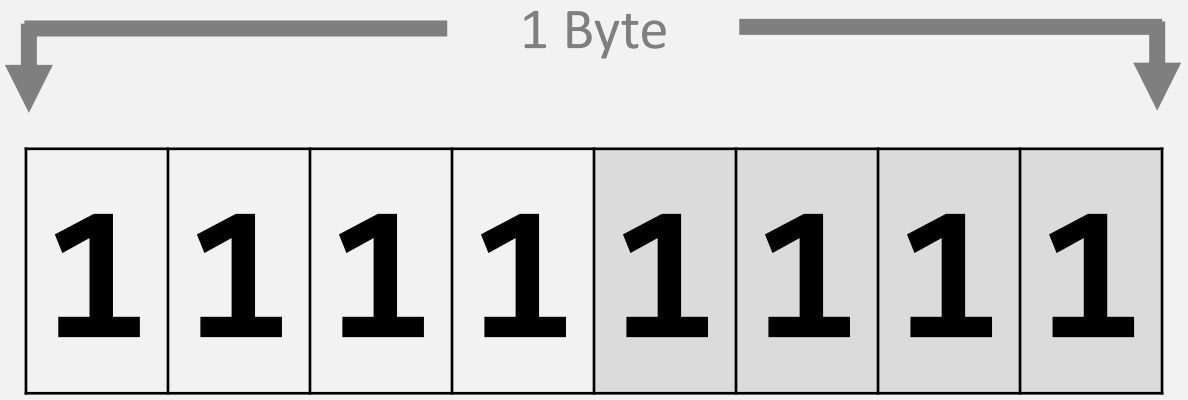
1991

**Hubble Telescope**

https://www.nbcnews.com/mach/video/5-amazing-discoveries-the-hubble-space-telescope-is-responsible-for-1217877571831

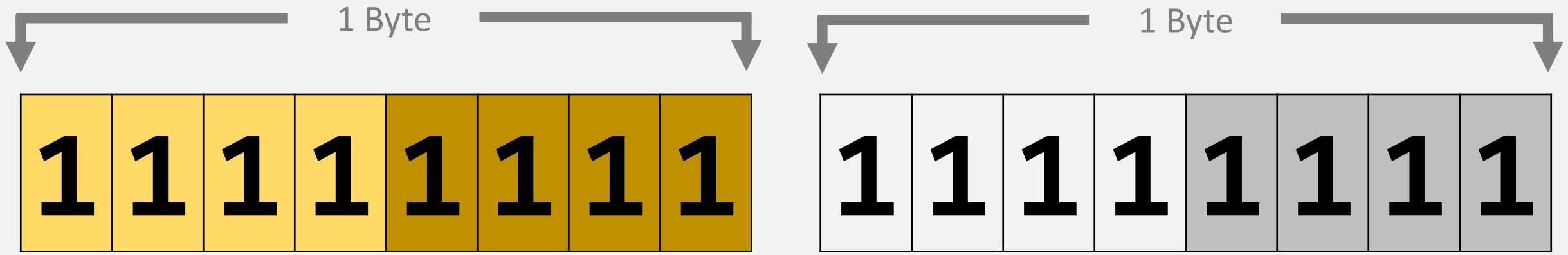

UNICODE

*“One byte is not enough”*




256 Characters

# “Let’s use two bytes per character”




65,536 Characters!

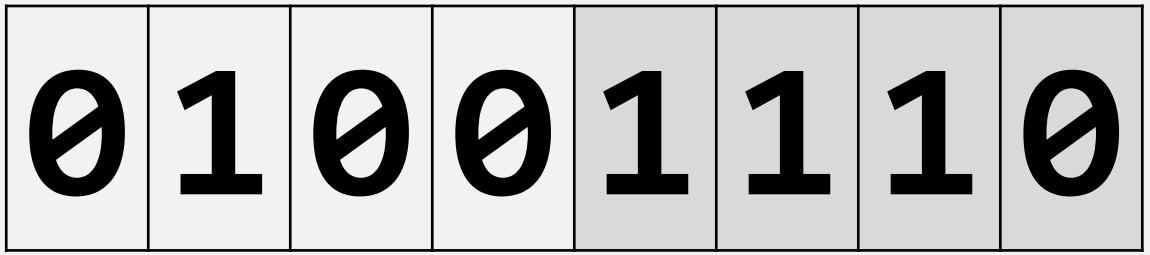
# “NL” in ASCII

N = ASCII 78

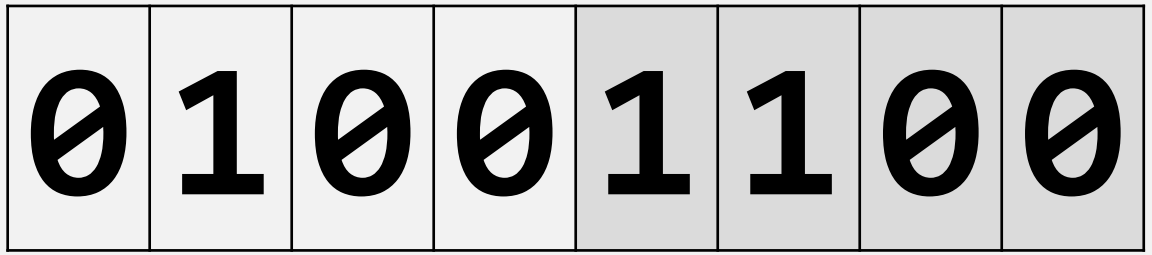
L = ASCII 76

# “NL” in ASCII

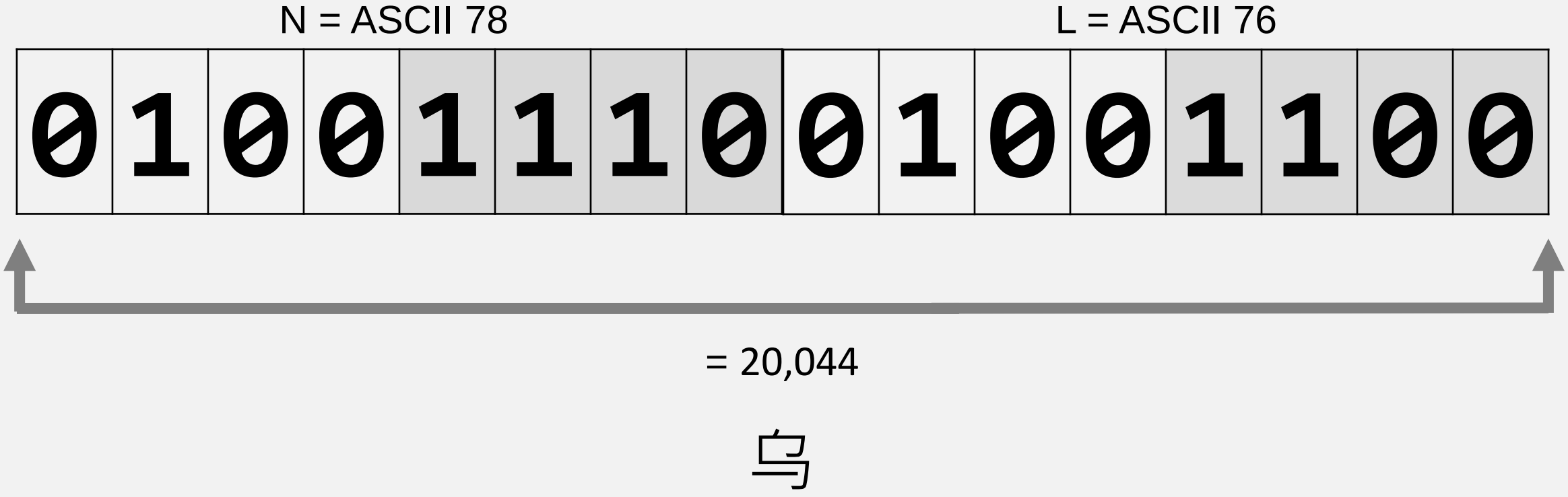
N = ASCII 78



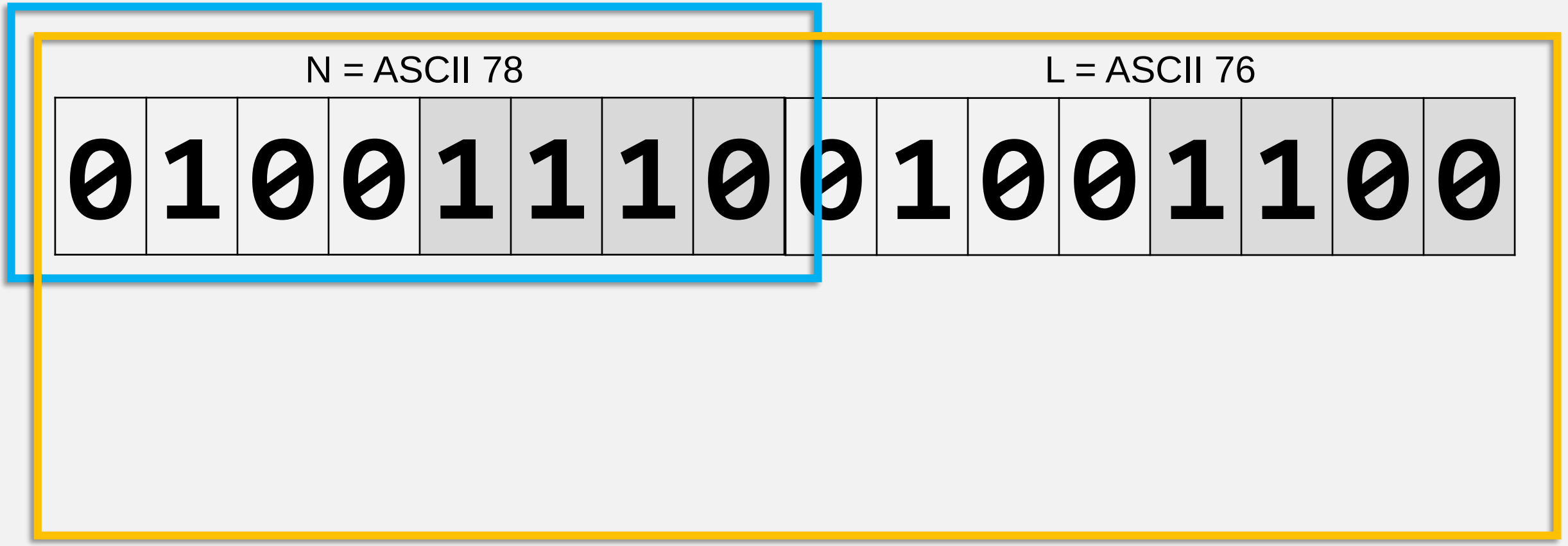

L = ASCII 76

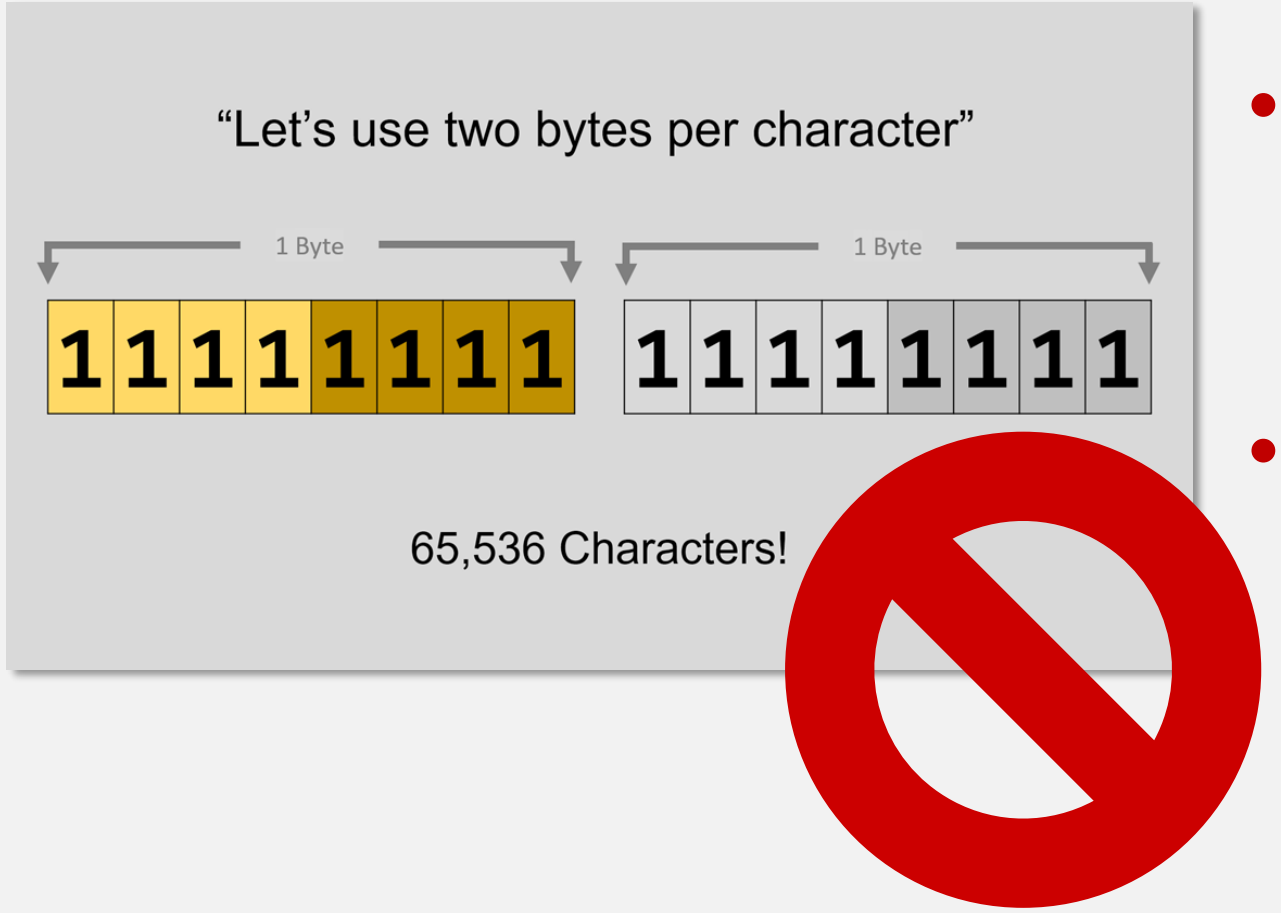


# “NL” in ASCII



# “NL” in ASCII






- Incompatible with existing documents
- Multiple representations for the same character

*“65,536 Characters are enough”*

# *“65,536 Characters are enough”*

- 1,056 Latin based characters
- 304 Cyrillic characters
- 384 Greek characters
- 304 Arabic characters
- 1,280 Indian writing systems
- 11,008 Korean characters
- 28,160 Chinese and Japanese characters

= 42,496+

- Miscellaneous scripts
- Mathematical symbols
- Scientific symbols
- Currency symbols


There's no emoticon for what I'm feeling!


UNICODE


- Consistent character representation
- Consistent and backwards-compatible character encoding
- Handle text used in most world’s writing systems
- A versioned and consistently improved standard


# 3

# Character Encodings

## Unicode Transformation Format

1. UTF-8 (8-32 bits per character)
2. UTF-16 (16 or 32 bits per character)
3. UTF-32 (Fixed 32 bits per character)


# 1,114,112

## Code Points

A code point is a numerical value that represents a code space, which can contain a single character, a formatting marker, a glyph, etc.

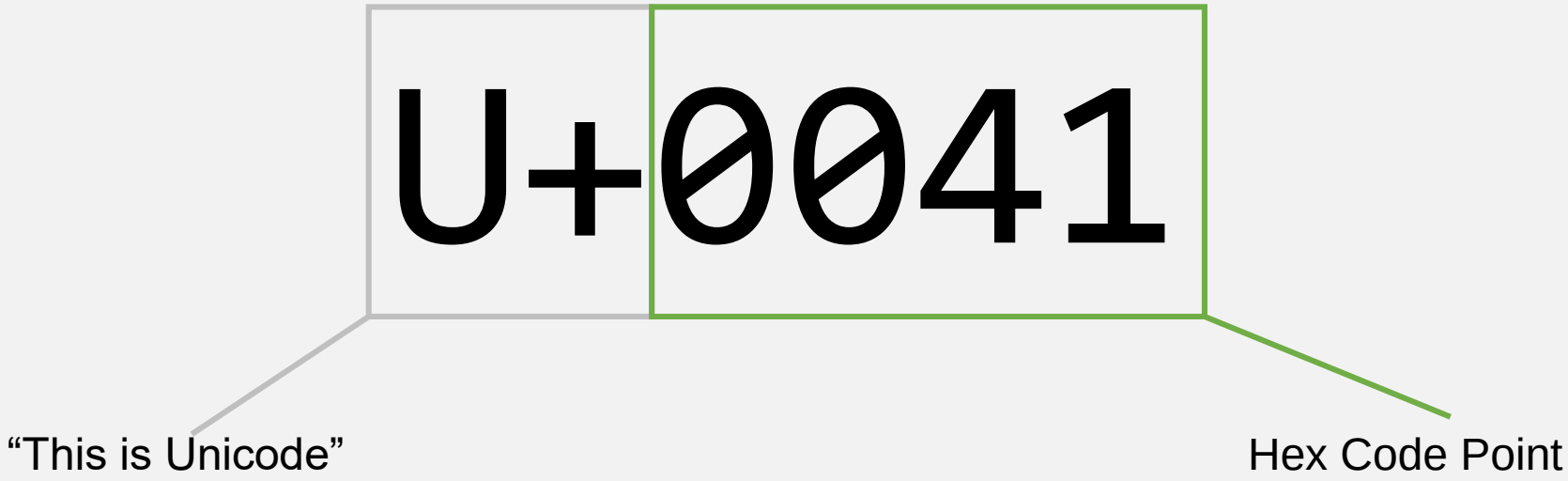

Unicode 12.1

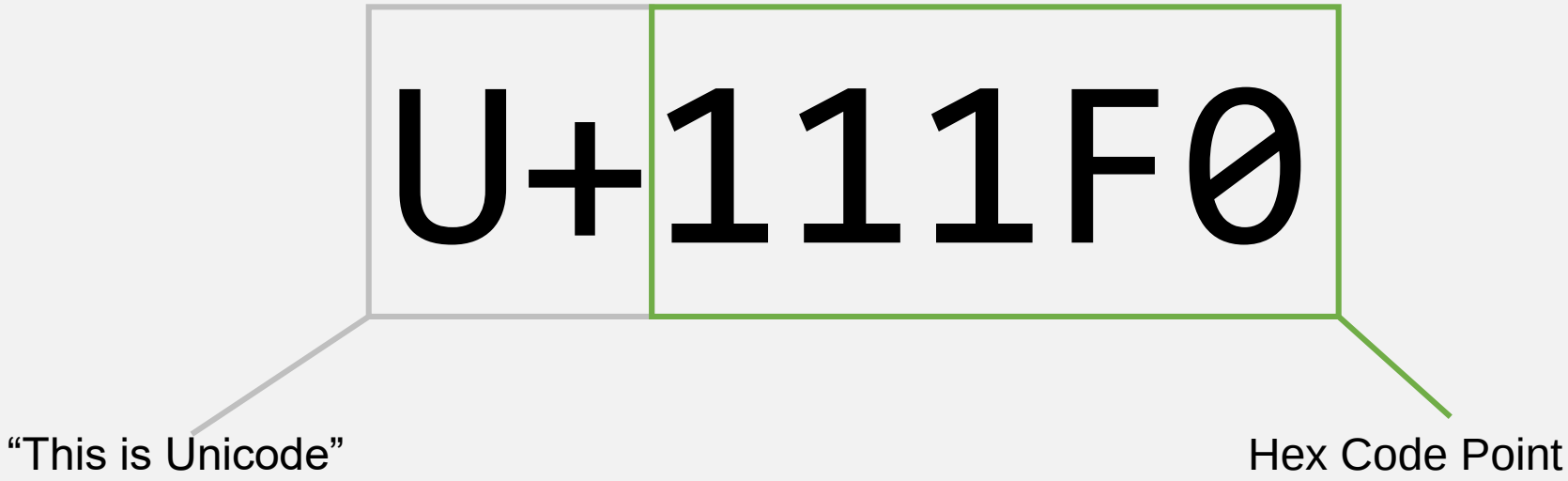
**137,994**

Characters

**150**

Writing Systems


U+0041
“This is Unicode”
Hex Code Point

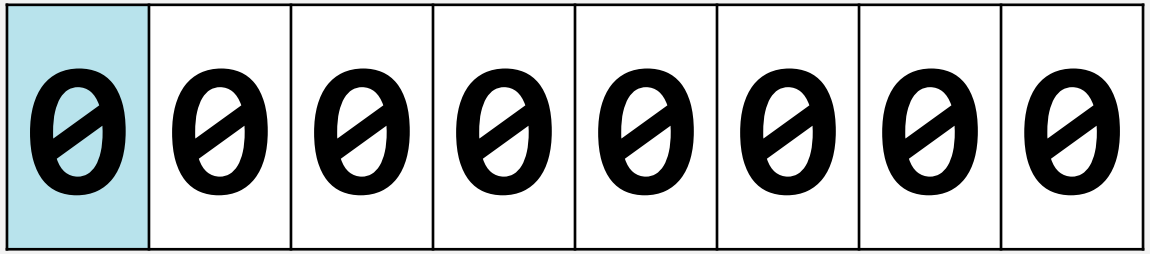

U+111F0
“This is Unicode”
Hex Code Point

# UTF-8

# UTF-8

U+0000 - U+007F

8 bits

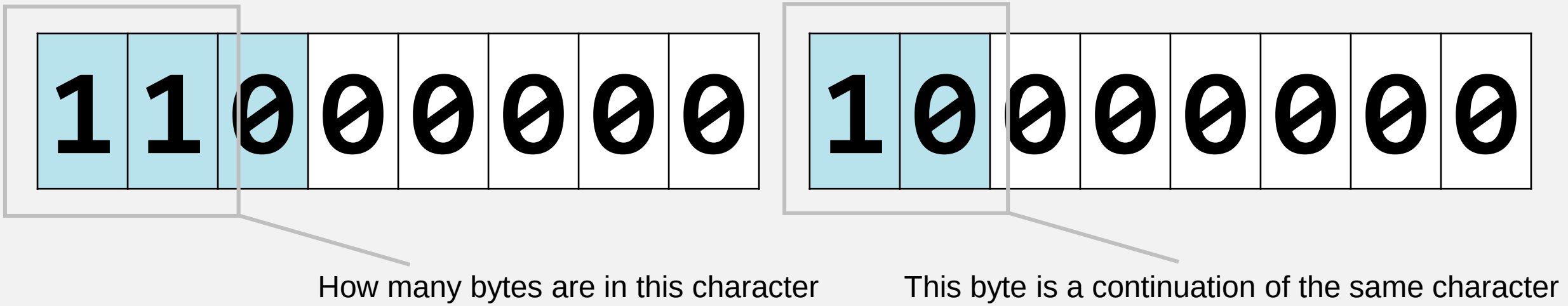



U+0000 (ASCII null)

# UTF-8

U+0080 - U+07FF

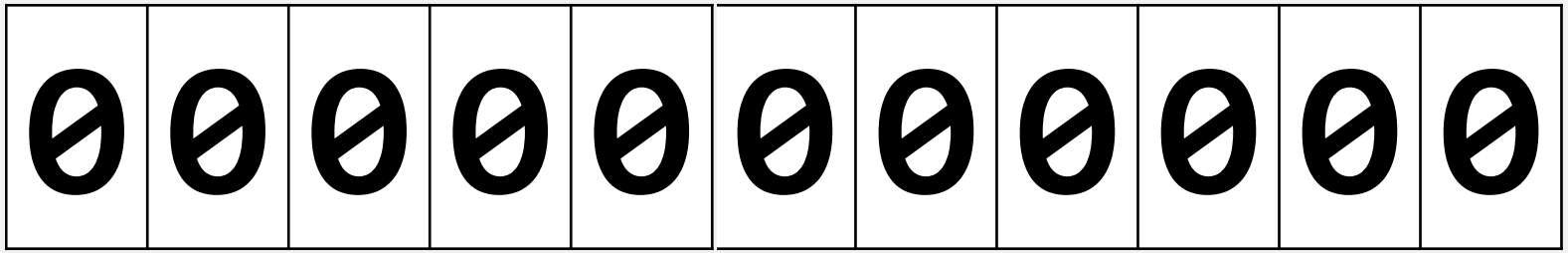
16 bits



# UTF-8

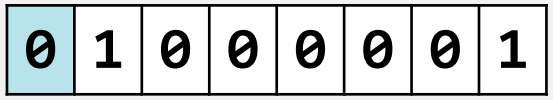
U+0080 - U+07FF

16 bits



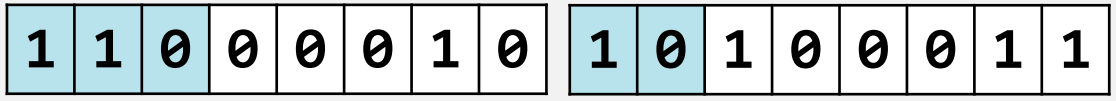
# UTF-8

U+0000-U+007F 1 byte - ASCII Compatible

A 

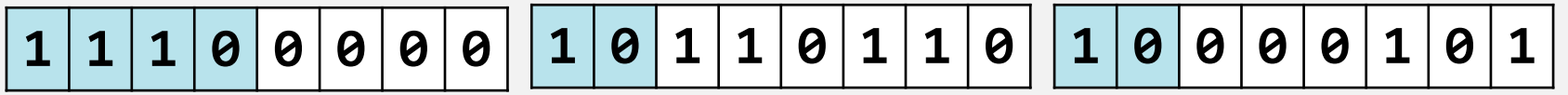
100 0001

U+0080-U+07FF 2 bytes

£ 

000 1010 1100

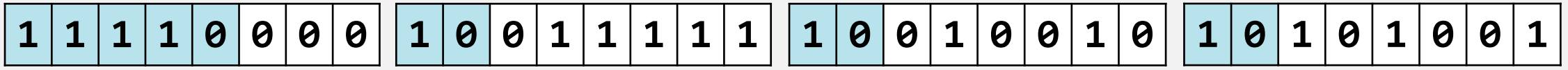
U+0800-U+FFFF 3 bytes

ह 

0000 1101 1000 0101

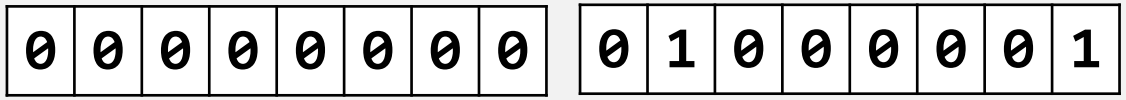
U+10000-U+10FFFF 4 bytes

0 0001 1111 0100 1010 1001

💩 

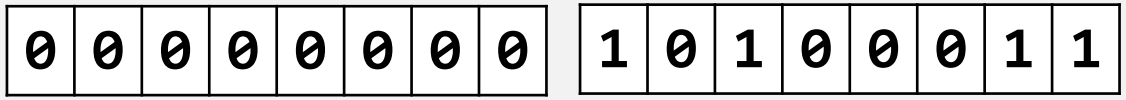
# UTF-16

U+0000-U+D7FF and U+E000 to U+FFFF 2 bytes

A 

0000 0000 0100 0001

U+0000-U+D7FF and U+E000 to U+FFFF 2 bytes

£ 

0000 0000 1010 0011

U+0000-U+D7FF and U+E000 to U+FFFF 2 bytes

ह 

0000 1101 1000 0101

U+010000 to U+10FFFF 4 bytes

0000 1111 0100 1010 1001

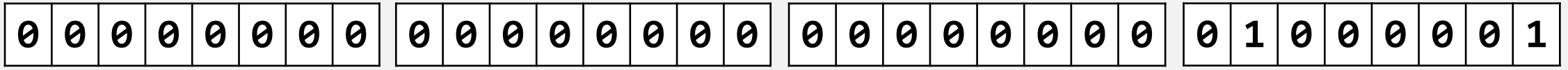
💩 

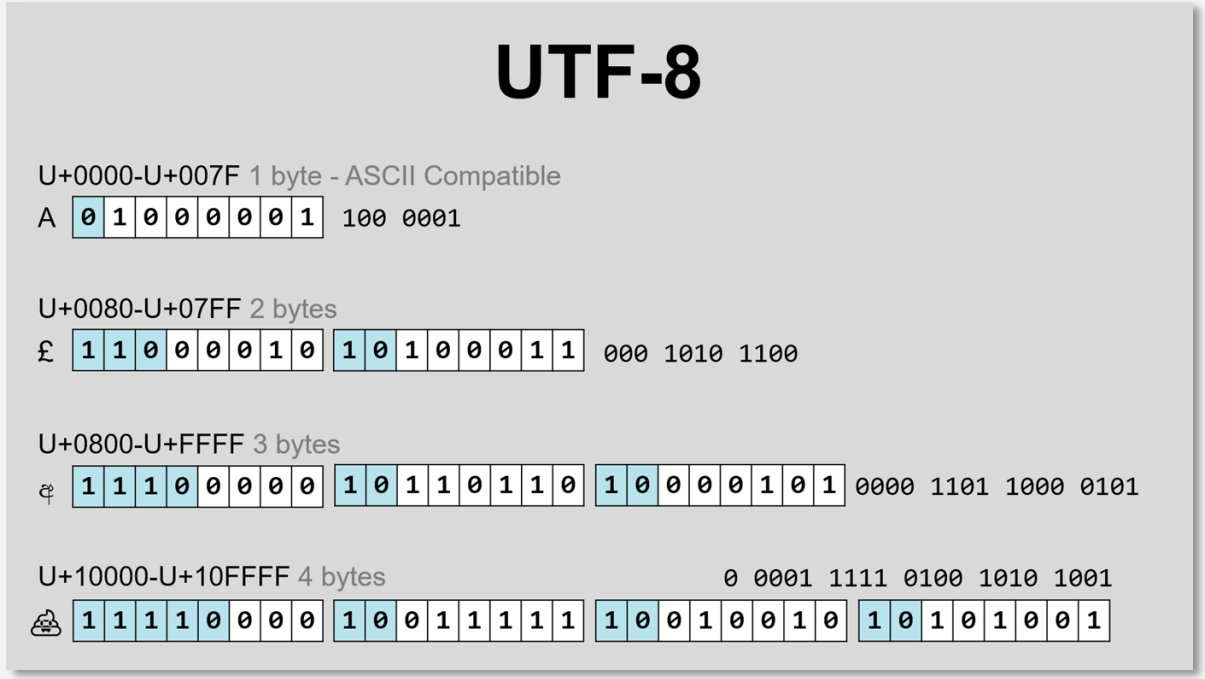
# UTF-32

U+0000 - U+10FFFF 4 bytes fixed

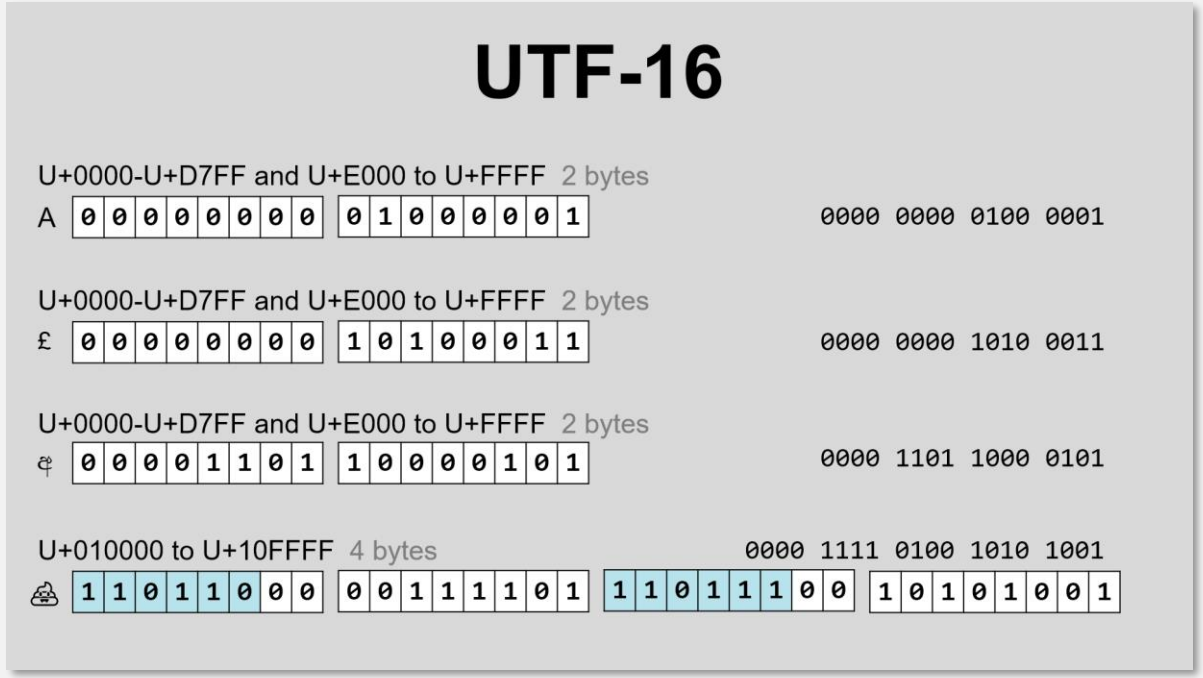
0000 0000 0000 0000 0000 0000 0100 0000

A

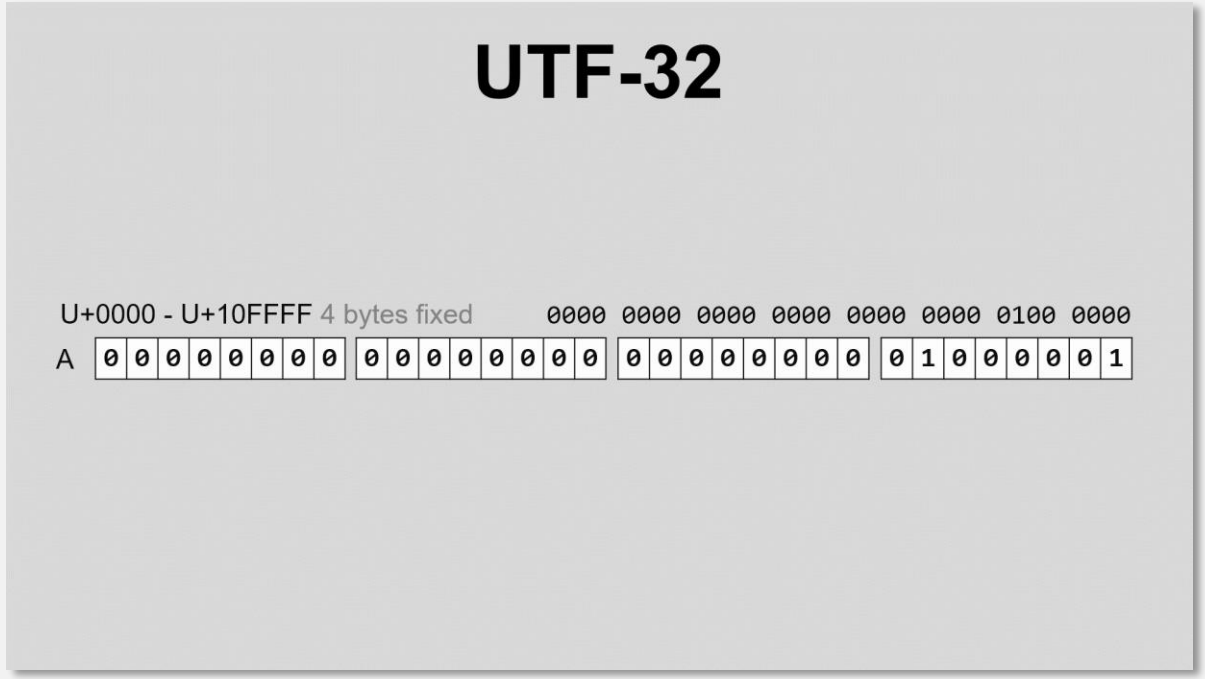





- Compatible with existing ASCII documents
- Does not waste storage space
- Character length is not fixed



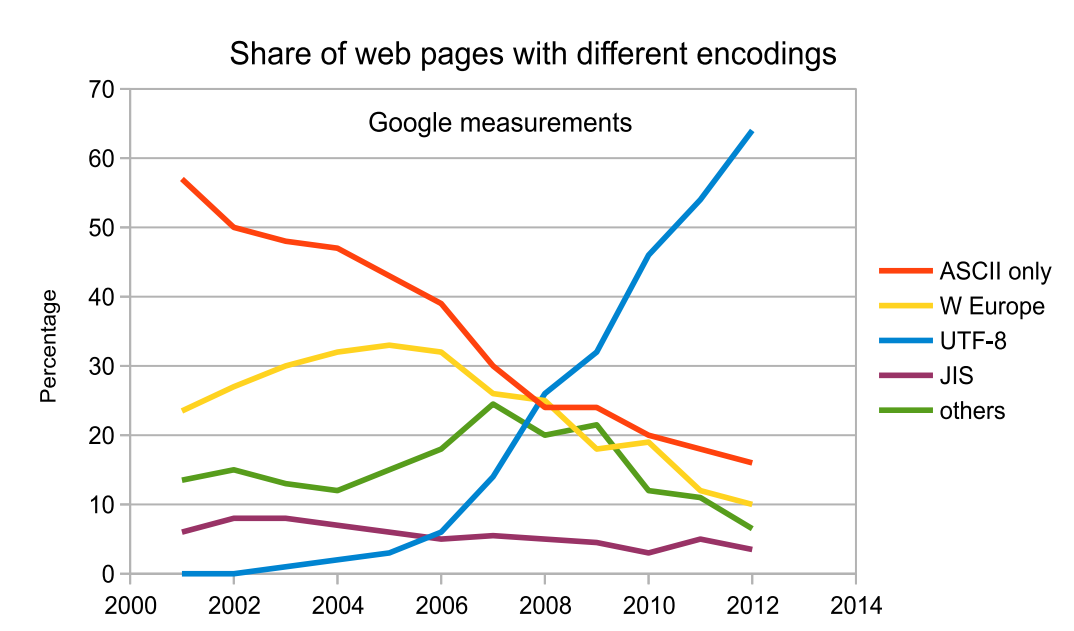

- Character length is not fixed, although it’s mostly 2 bytes
- Wastes memory
- Incompatible with ASCII and other character encodings




- Each character is fixed-length

- Wastes memory

- Incompatible with ASCII and other character encodings




https://googleblog.blogspot.com/2012/02/unicode-over-60-percent-of-web.html

# UTF-8

U+0000-U+007F 1 byte - ASCII Compatible

A 0 1 0 0 0 0 0 1 100 0001

U+0080-U+07FF 2 bytes

£ 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 1 000 1010 1100

U+0800-U+FFFF 3 bytes

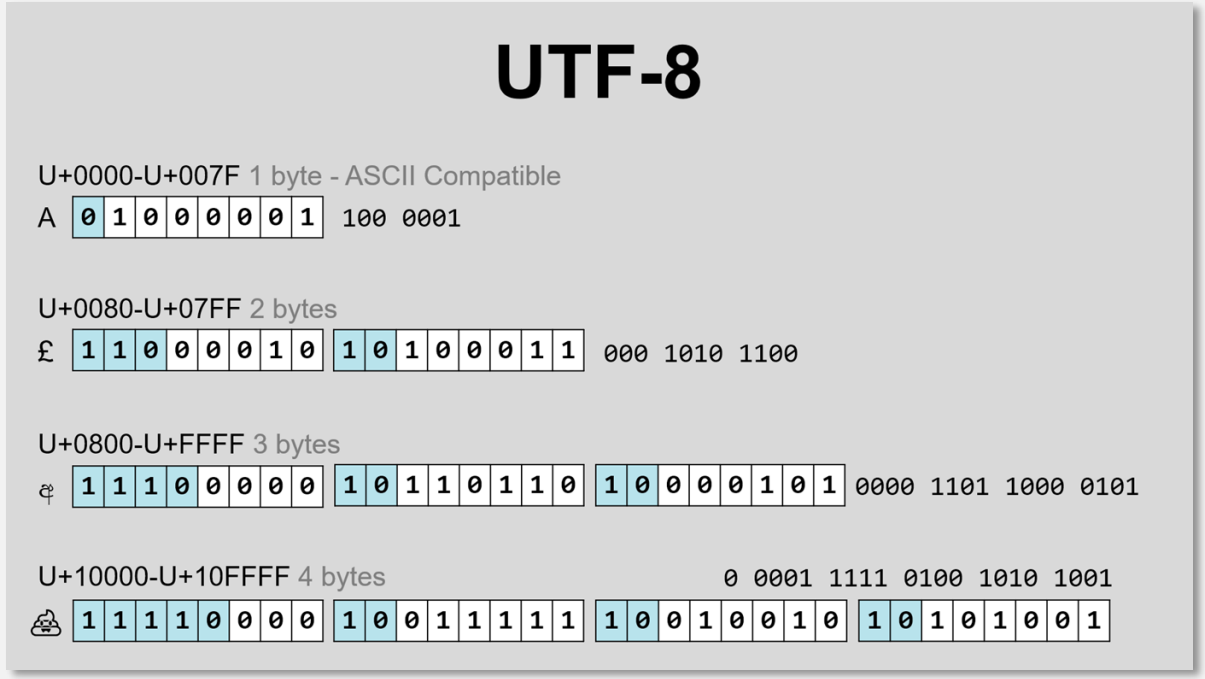
ह 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0000 1101 1000 0101

U+10000-U+10FFFF 4 bytes 0 0001 1111 0100 1010 1001

1 1 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1

# 93.5%

Web Pages



# MANDATED

WHATWG

“for all things”

## UTF-8

U+0000-U+007F 1 byte - ASCII Compatible

A `0 1 0 0 0 0 0 1` 100 0001

U+0080-U+07FF 2 bytes

£ `1 1 0 0 0 0 1 0` `1 0 1 0 0 0 1 1` 000 1010 1100

U+0800-U+FFFF 3 bytes

ह `1 1 1 0 0 0 0 0` `1 0 1 1 0 1 1 0` `1 0 0 0 0 1 0 1` 0000 1101 1000 0101

U+10000-U+10FFFF 4 bytes 0 0001 1111 0100 1010 1001

💩 `1 1 1 1 0 0 0 0` `1 0 0 1 1 1 1 1` `1 0 0 1 0 0 1 0` `1 0 1 0 1 0 0 1`

# Unicode for Developers

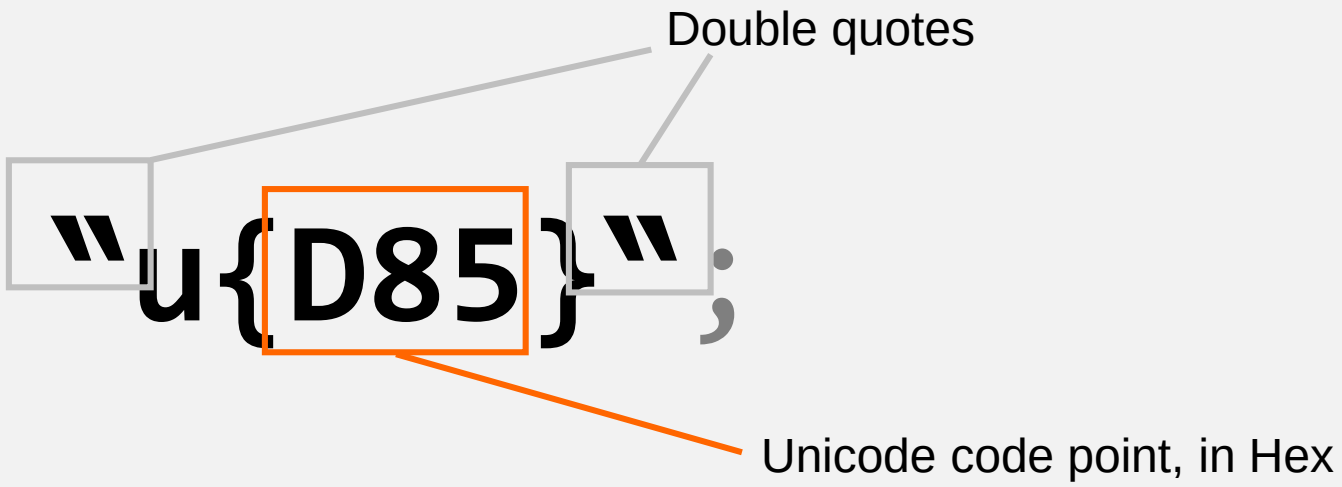
A.K.A The Juicy Bits

# How do you type Unicode characters in PHP?

# How do you type Unicode characters in PHP?

```
$var = 'ඞ';
```

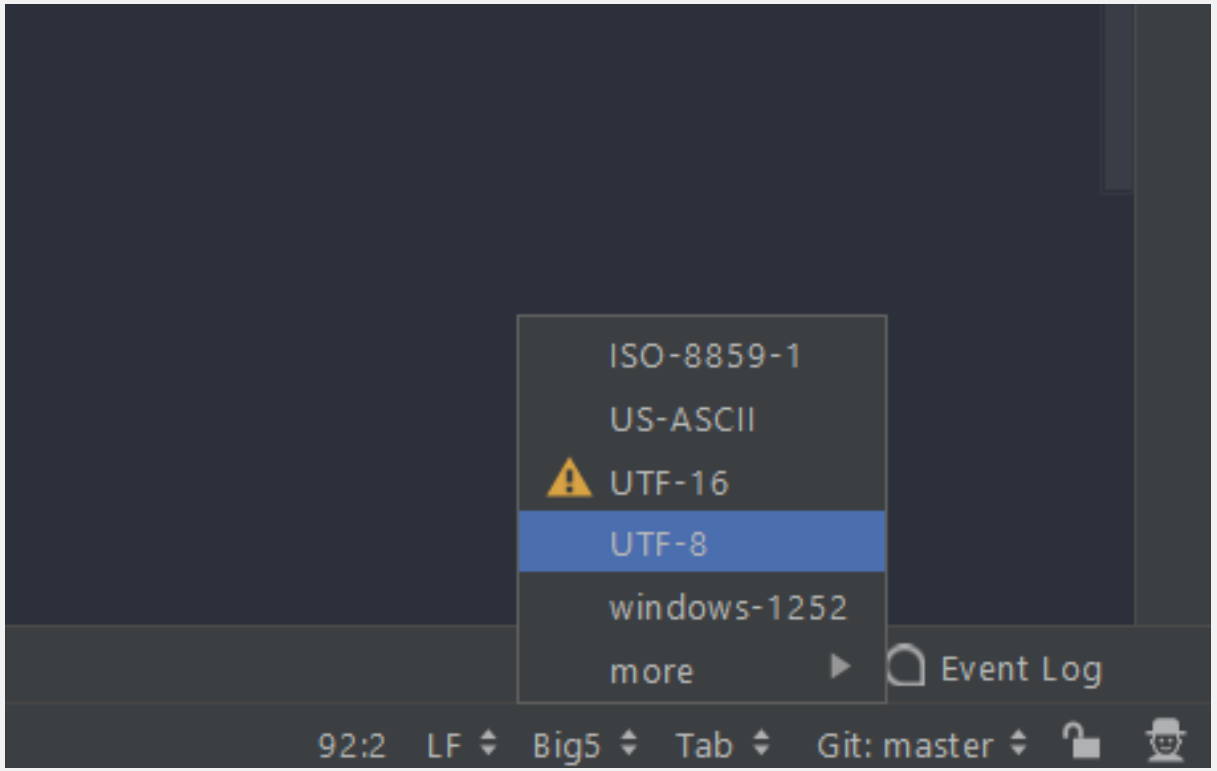
```
$var = "\u{D85}";
```



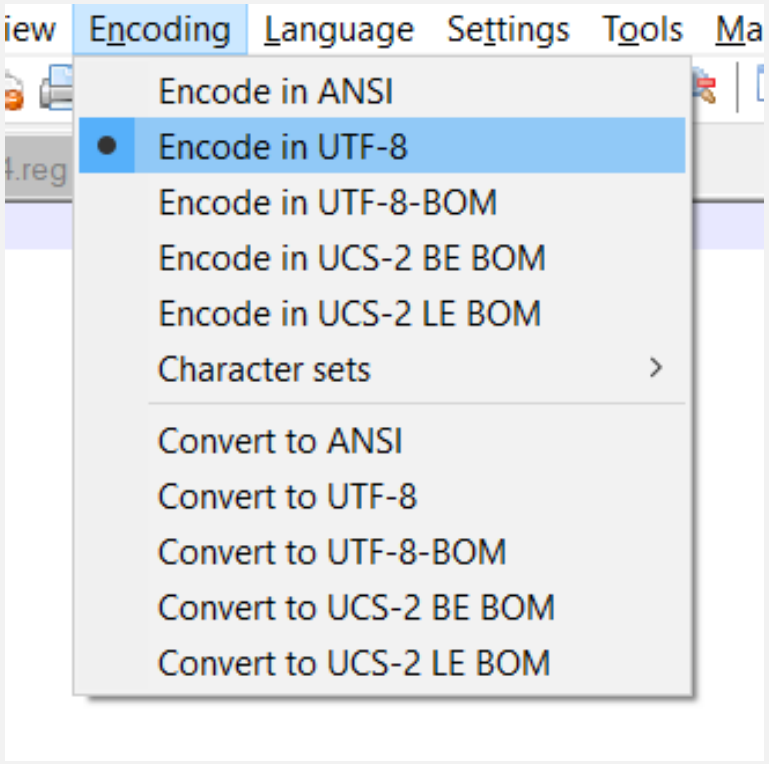
# Always save PHP files as UTF-8

# Always save PHP files as UTF-8

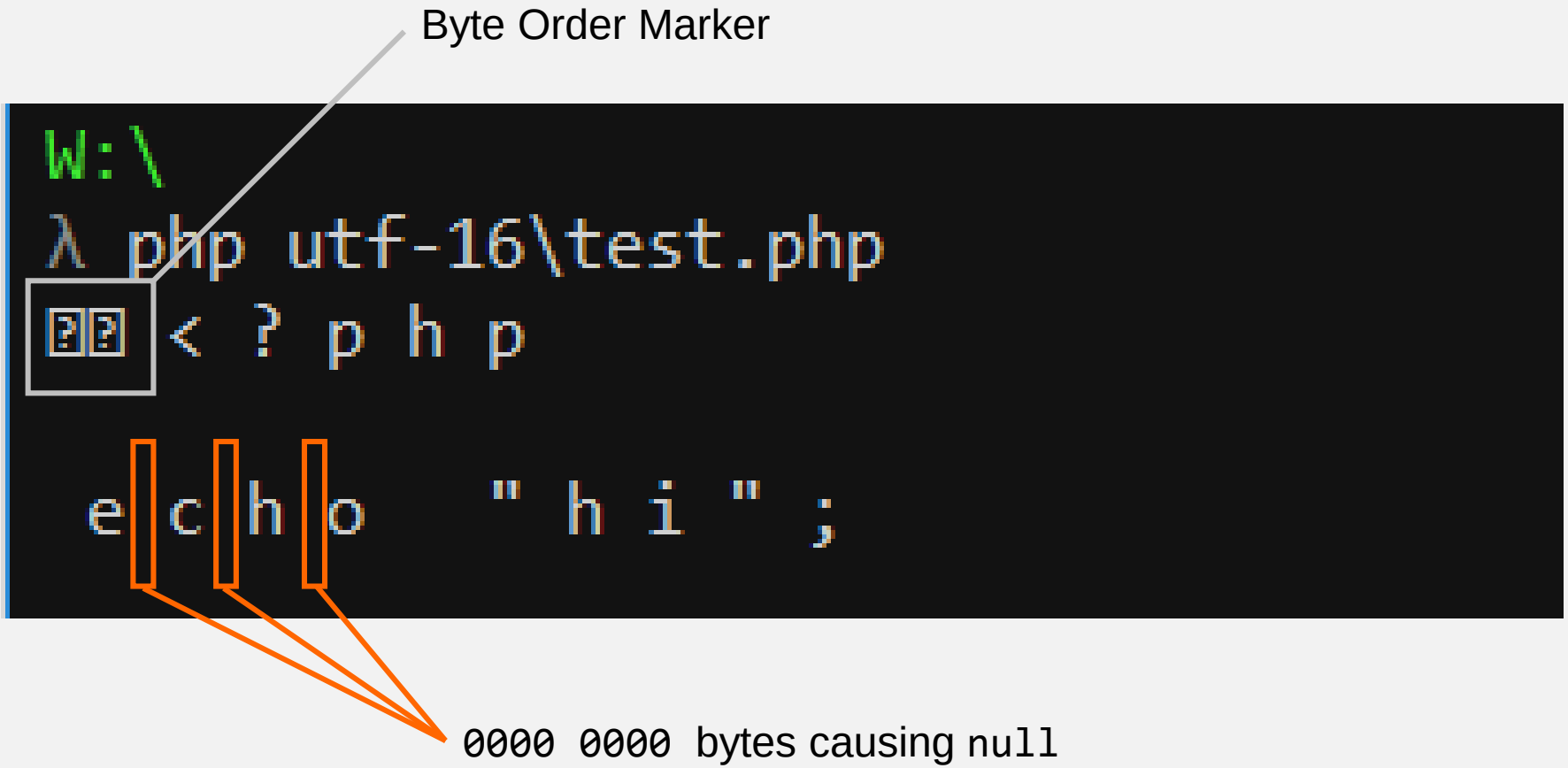
PHPStorm




Notepad++



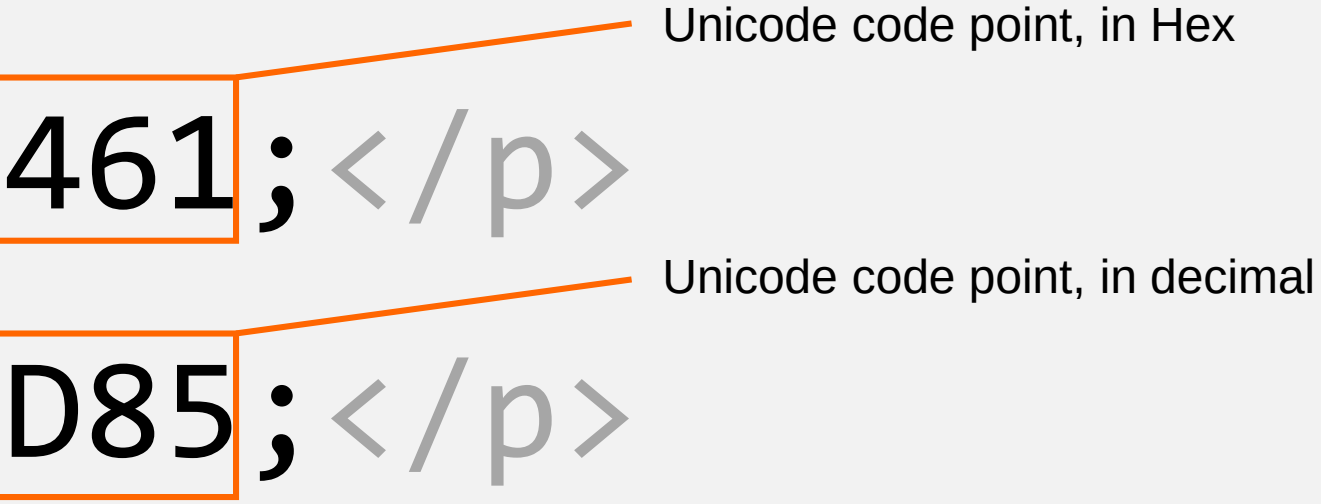
# Always save PHP files as UTF-8



# How do you type Unicode characters in HTML?

# How do you type Unicode characters in HTML?

```
<p>ఫ</p>
<p>&#3461;</p>
<p>&#xD85;</p>
```



# Tell browsers you are using UTF-8

# Tell browsers you are using UTF-8

```
++ header('Content-Type: text/html; charset=utf-8');
```

# PHP considers every byte as a single character

# PHP considers every byte as a single character

String Length

```
strlen('€')
= 3
```

**Expected**

1

# PHP considers every byte as a single character

Sub-string

```
substr('€10,25', 1, 5);
= b",¬10,"
```

**Expected**

```
'10,25'
```

# PHP considers every byte as a single character

String Array Access

```
echo '😍😢😛'[2];
```

= 

**Expected**

😛

# Use PHP Multi-Byte extension (`mb_*`)

# Use PHP Multi-Byte extension (mb_*)

String Length

```
-- strlen('€')

++ mb_strlen('€')

    = 1
```

# Use PHP Multi-Byte extension (mb_*)

Sub-string

```
-- substr('€10,25', 1, 5);
++ mb_substr('€10,25', 1, 5);
    = '10,25'
```

# Use PHP Multi-Byte extension (mb_*)

String Array Access

```
-- '😍😢😛'[2];
++ mb_substr('😍😢😛', 2, 1);
```

# Use PHP Multi-Byte extension (`mb_*`)

For many other string manipulation functions

- `mb_strtoupper`
- `mb_strtolower`
- `mb_strlen`
- `mb_strops`
- `mb_strrpos`
- `mb_stripos`
- `mb_strripos`
- `mb_strstr`
- `mb_strrchr`
- `mb_stristr`
- `mb_strrichr`
- `mb_substr_count`
- `mb_substr`
- …

# Do not overload mb_* functions

# Do not overload mb_* functions

**php.ini**

`-- mbstring.func_overload`

`mb_*` function overloading is supposed to Overload the standard string manipulation functions.

To ensure consistency, this is discouraged and is deprecated in PHP 7.2.

# mb_* functions are slow ?

# mb_* functions are slow ?

```
strlen('€’)
strlen('€€€€€……1000')
strlen('€€€€€……10000’)
```

```
mb_strlen('€’)
mb_strlen('€€€€€……1000’)
mb_strlen('€€€€€……100000')
```

x 1 million

$T_1$ = 0.322 S
$T_2$ = 0.413 S
$T_3$ = 0.524 S

$T_1$ = 1.135 S
$T_2$ = 58.709 S
$T_3$ = 523.173 S

mb_* functions *are* slower, with complexity **O(n)**

# `utf8_encode/decode` functions are not magic

# `utf8_encode/decode` functions are not magic

`utf8_encode` and `utf8_decode` function names can be misleading.

They **do not** fix PHP's *one byte is one character* behavior

# `utf8_encode/decode` functions are not magic

`utf8_encode`

ISO 8859-1 -> UTF8

`utf8_decode`

UTF8 -> ISO 8859-1

# Detect and encode incoming data in UTF-8

# Detect and encode incoming data in UTF-8

```
$encoding = mb_detect_encoding($str, mb_list_encodings());
$str = mb_convert_encoding($str, 'UTF-8', $encoding);
```

# Regular Expressions with Unicode

# Regular Expressions with Unicode

On a single Unicode code point

```
preg_match('/\x{0041}/u', $str, $matches);
```

# Regular Expressions with Unicode

On a Unicode code point range

```
preg_match('/[\x{0041}-\x{0045}]/u', $str, $matches);
```

# Regular Expressions with Unicode

On a named Unicode group

```
preg_match('/[\p{XY}/u', $str, $matches);
```

*Find matches for XY group*

```
preg_match('/[\P{XY}/u', $str, $matches);
```

Find inverse matches for XY Group

# Regular Expressions with Unicode

On a named Unicode group

```
preg_match('/[\p{Sc}/u', $str, $matches);
```

€12, $15

# Regular Expressions with Unicode

On a named Unicode group

```
preg_match('/[\p{Sinhala}/u', $str, $matches);
```

A්K

# Regular Expressions with Unicode

On a named Unicode group

```
preg_match('/[\P{Sinhala}/u', $str, $matches);
```

A්K

# Unicode with MySQL

# Unicode with MySQL

What is **Charset**?

A set of characters that are supported and allowed in the storage system.

# Unicode with MySQL

What is **Collation**?

Rules for sorting and handling of the characters in the given charset.

# Unicode with MySQL

## Charsets and Collation

A given charset can contain one or more collations.

# Unicode with MySQL

## MySQL Configuration

**my.cnf**

```
[client]
default-character-set = utf8mb4

[mysql]
default-character-set = utf8mb4

[mysqld]
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci
```

# Unicode with MySQL

Connecting to a database

```
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
$pdo = new PDO($dsn, $user, $pass, $options);
```

# Unicode with MySQL

## utf8 vs utf8mb4

Older MySQL versions only had **`utf8`** charset, which only supported 3 bytes per character, which left Emojis, Chess piece icons, etc. out.

**`utf8mb4`** fully supports all Unicode characters

# Unicode with MySQL

## Convert charset and collation to utfmb4

```
ALTER DATABASE database_name CHARACTER SET = utf8mb4
COLLATE = utf8mb4_unicode_ci;

# For each table.
ALTER TABLE table_name CONVERT TO CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

# For each column
ALTER TABLE table_name CHANGE column_name
VARCHAR(191) CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

# Unicode with MySQL

## MySQL max index size

MySQL supports a maximum index size (`InnoDB`, `MyISAM`, etc.) of `768` bytes. This previously allowed `255` characters to be indexed.

With **`utf8mb4`**, we need to allocate **4 bytes** per character (at worse case), Which gives us a less index size for VARCHAR columns.

# Unicode + Security

# Unicode + Security

- PayPal attacks: Visually similar characters enabling homograph attacks.
- Malicious fonts overriding the glyphs.
- Text comparison considerations.
- `fopen`/`fseek`/`fwrite` operations take byte parameters. Not characters


I ? Unicode

# Further Resources

# Further Resources

- https://unicode.org
- https://www.compart.com/en/unicode/
- Polyglot Gathering
- https://www.regular-expressions.info/unicode.html
- https://www.php.net/manual/en/class.transliterator.php
- https://www.php.net/manual/en/regexp.reference.unicode.php
- https://www.toptal.com/php/a-utf-8-primer-for-php-and-mysql
- https://twitter.com/FakeUnicode
- Ken Thompson quotes
- https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-.../
- https://www.youtube.com/watch?v=YDYEEC2rPmg
- https://www.youtube.com/watch?v=MijmeoH9LT4
- https://dev.mysql.com/doc/refman/5.5/en/charset-unicode-conversion.html
- https://dev.mysql.com/doc/refman/5.5/en/charset-unicode-upgrading.html
- http://utf8everywhere.org/

# Questions 

No question is too small. No question is stupid.

**Ruby Lounge (upstairs)**

**@Ayeshlive** ayesh@ayesh.me

https://ayesh.me/talk/Unicode

# THANK YOU

arigatô
paldies
dziękuję
Ďakujem
tak
mahalo
děkuji
kop khun
diolch
dankie
köszönöm
감사합니다
хвала
shukran
tänan
a dank
gràcies
ngiyabonga
Баярлалаа
dhanyavād
Дякую
Благодарам
ευχαριστώ
спасибо
благодаря
tack
grazie
Ǹh’gōi
Dank u
gracias
merci
ස්තුතියි
mulţumesc
ačiū
תודה.
takk
nandri
danke
faleminderit
Xièxiè
choukrane
teşekkür ederim
obrigado
kiitos
Շնորհակալություն
terima kasih
hvala
grazzi


Unicode + PHP
Ayesh Karunaratne