文字列をイテレーションする

Iterate over "elements" of a string.

第138回 PHP勉強会@東京

2019年5月29日 #phpstudy

お前誰よ

- うさみけんた（@tadsan） / Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- ピクシブ株式会社 pixiv運営本部
- Emacs Lisper, PHPer
  - Emacs PHP Modeのメンテナ引き継ぎました
  - 好きなリスプはEmacs Lispです
- Qiitaに記事を書いたり変なコメントしてるよ

pixiv

宣伝

PHP Mode for GNU Emacs

This is a major mode development project to support PHP coding in GNU Emacs. This fork builds on the work of:

1. Turadg Aleahmad (Original Author)
2. Aaron S. Hawley
3. Lennart Borgman
4. Eric James Michael Ritz
5. Syohei Yoshida

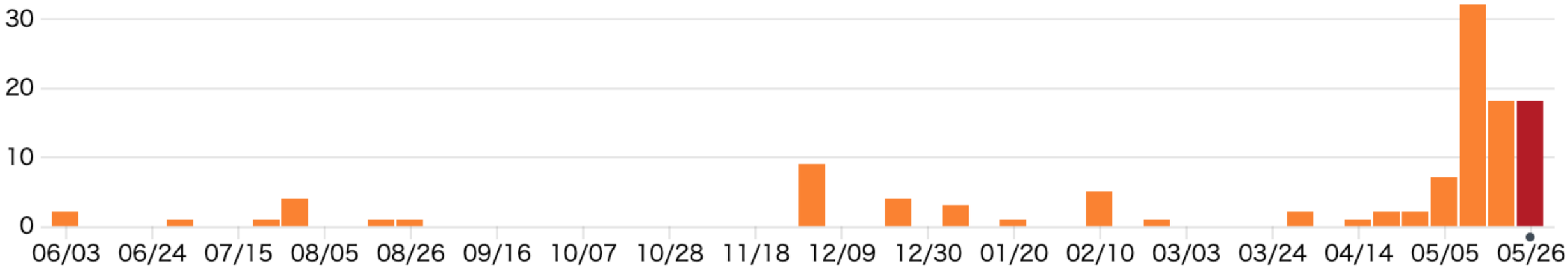
All contributors listed below improved PHP Mode as well.

The current maintainer is:

1. USAMI Kenta (@zonuexe)

Please submit any bug reports or feature requests by creating issues on the GitHub page for PHP Mode. Alternatively you may also request features via the FeatHub page for the entire PHP suite for GNU Emacs.

Installation


30
20
10
0
06/03
06/24
07/15
08/05
08/26
09/16
10/07
10/28
11/18
12/09
12/30
01/20
02/10
03/03
03/24
04/14
05/05
05/26

先に

今回話す内容は
平成最後のLT大会で
発表した内容に由来

平成時代に憧れてた
あのエディタを実装してみる

Implement the text editor I admired in Heisei era

新元号決定！平成最後のLT大会＆PARTY
2019年4月30日 #engineers_lt

今回話す実装は
既にライブラリ化
しました

Bag2 String Iterator (`\Bag2\iter\string`)

Functions for to iterate string/bytes.

Functions

`each_byte`

Use this function specifically to iterate byte by byte.

NOTICE: In UTF-8, one character is not one byte.

```php
<?php

use function Bag2\iter\string\each_byte;

$string = "abcdef";

foreach (each_byte($string) as $s) {
    echo $s, PHP_EOL;
}
// a
// b
// c
// d
// e
```

おさらい

イテレータとは何か

任意の処理を
繰り返しの構造
に落とし込める

繰り返し？

みんなだいすき

foreach

どうするの？

Iterator

インターフェイス

Iterator インターフェイス

(PHP 5, PHP 7)

はじめに

外部のイテレータあるいはオブジェクト自身から反復処理を行うためのインターフェイスです。

インターフェイス概要

```
Iterator extends Traversable {

    /* メソッド */
    abstract public current ( void ) : mixed
    abstract public key ( void ) : scalar
    abstract public next ( void ) : void
    abstract public rewind ( void ) : void
    abstract public valid ( void ) : bool
}
```

IteratorAggregate インターフェイス

(PHP 5, PHP 7)

はじめに

外部イテレータを作成するためのインターフェイスです。

インターフェイス概要

```
IteratorAggregate extends Traversable {

    /* メソッド */
    abstract public getIterator ( void ) : Traversable
}
```

イテレータ

- AppendIterator
- ArrayIterator
- CachingIterator
- CallbackFilterIterator
- DirectoryIterator
- EmptyIterator
- FilesystemIterator
- FilterIterator
- GlobIterator
- InfiniteIterator
- IteratorIterator
- LimitIterator
- MultipleIterator
- NoRewindIterator
- ParentIterator
- RecursiveArrayIterator
- RecursiveCachingIterator
- RecursiveCallbackFilterIterator
- RecursiveDirectoryIterator
- RecursiveFilterIterator
- RecursiveIteratorIterator
- RecursiveRegexIterator
- RecursiveTreeIterator
- RegexIterator

SPL にはイテレータが用意されており、オブジェクトを反復処理することができます。

ジェネレータ

関数で
イテレータを
つくれる

時間がないので割愛

このあと

さんざん出ます

さて本題

文字列をイテレーションする

Iterate over "elements" of a string.

第138回 PHP勉強会@東京

2019年5月29日 #phpstudy

問題です

入力した文字列を
1文字ごとに改行
区切りで出力せよ

簡単じゃん

```php
$string = "abcde";

$length = \strlen($string);

for ($i = 0; $i < $length; $i++) {
    echo $string[$i], PHP_EOL;
}
```

```
>>> >>> _> g
>>>
a
b
c
d
e
```

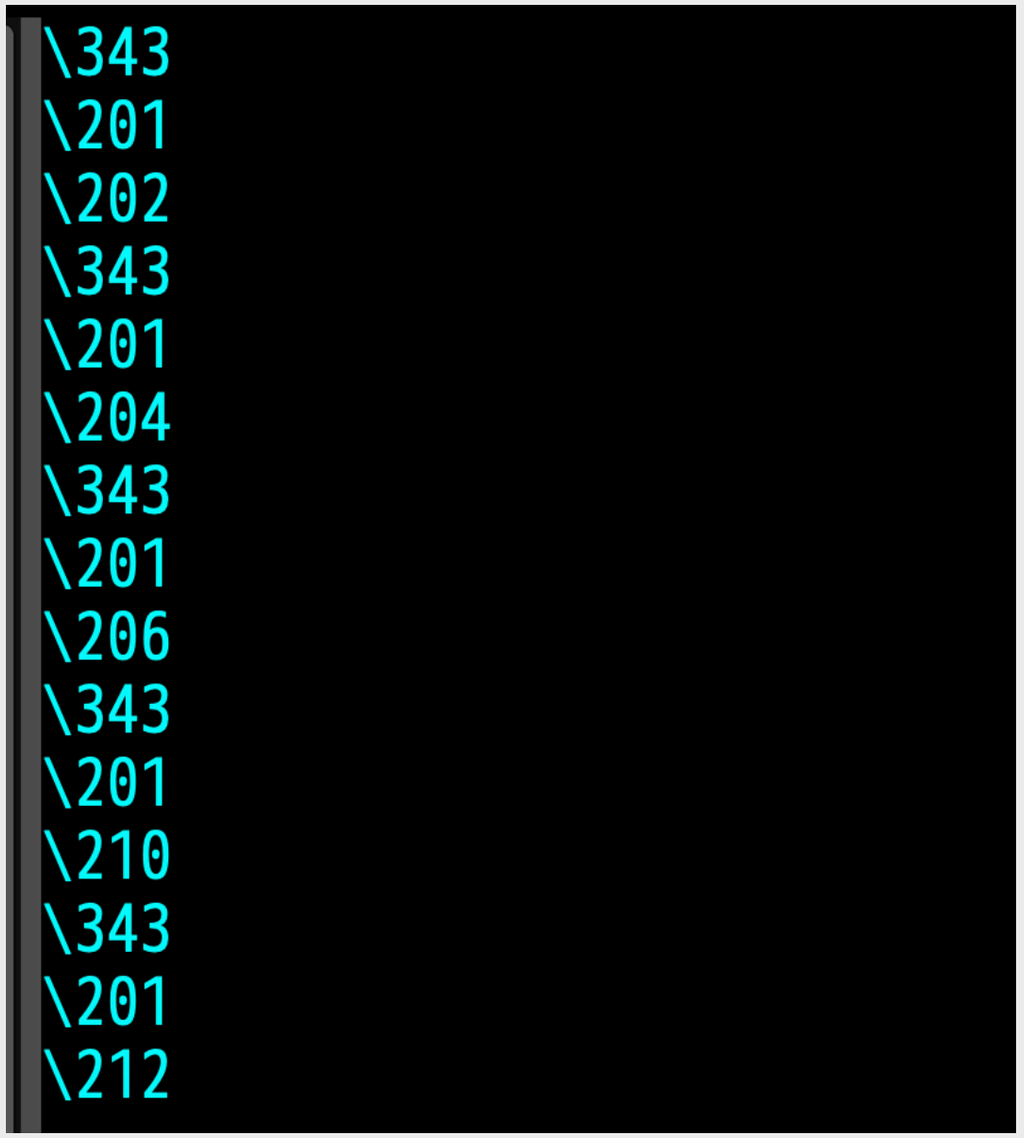
簡単でしょ？

```php
$string = "あいうえお";

$length = \strlen($string);

for ($i = 0; $i < $length; $i++) {
    echo $string[$i], PHP_EOL;
}
```

本当に？


\343
\201
\202
\343
\201
\204
\343
\201
\206
\343
\201
\210
\343
\201
\212

(どう表示されるかは環境による)

. . . . . .

どうしてこんな
ことがおこるのか

前提1:

PHPのstringは「文字」を知らない

stringは

ただのバイト列

```php
$s = 'abc';
echo $s[0];
```

これは先頭1文字ではなく
1バイトを取り出す

逆に言うとstring
にはどんなバイナ
リも格納できる

例:

```
<?php
$s = file_get_contents('flower.png');
header('Content-Type: image/png');
echo $s;
```

mb関数が

あるのでは？

mb_chr

(PHP 7 >= 7.2.0)

mb_chr — Get a specific character

説明

```
mb_chr ( int $cp [, string $encoding ] ) : string
```

mb関数はencodingを常に
指定しないと(環境依存せず)
確実に処理できない

PHP5.6以降では
default_charsetの
デフォルト値が"UTF-8"に、
mbstring.internal_encodingが
非推奨になったのは嬉しい

それはencodingを明示的に
指定しなくても安全に処理
できることを意味しない

前提2:

UTF-8は

可変長バイト

ひとつの符号位置を

表すために

1〜4バイト

まだ日本語文字を
「2バイト文字」と
呼んだりしませんね？

Unicodeより
前の時代

Shift_JISでは
ガ(2バイト)
ガ（1バイト+1バイト）

この頃は半角カナ
で情報量が半分に
なることもあった

UTF-8では
ガ(3バイト)
ガ (3バイト+3バイト)

Unicode(UTF-8)で
表現すると
半角カナは単純に
バイト数が倍になる

たいへんですね

その上で
Unicode時代の今日
何を「文字」と
呼ぶかを考える

1. バイトごと
2. コードポイントごと
3. 書記素クラスタごと

バイトごと

```php
$string = "abcde";

$length = \strlen($string);

for ($i = 0; $i < $length; $i++) {
    echo $string[$i], PHP_EOL;
}
```

これがまさに
バイトごとの
イテレーション

これを
ジェネレータに
変換

Bag2 String Iterator (`\Bag2\iter\string`)

Functions for to iterate string/bytes.

Functions

`each_byte`

Use this function specifically to iterate byte by byte.

NOTICE: In UTF-8, one character is not one byte.

```php
<?php

use function Bag2\iter\string\each_byte;

$string = "abcdef";

foreach (each_byte($string) as $s) {
    echo $s, PHP_EOL;
}
// a
// b
// c
// d
// e
```

```php
function each_byte(string $string)
{
    $length = \strlen($string);

    for ($i = 0; $i < $length; $i++) {
        yield $string[$i];
    }
}
```

ライブラリとしては
「foreachできるオブジェクト」
(=ジェネレータ)として提供

そうすることで
汎用的に利用で
きるようになる

```php
<?php

use function Bag2\iter\string\each_byte;

$string = "abcdef";

foreach (each_byte($string) as $s) {
    echo $s, PHP_EOL;
}
// a
// b
// c
// d
// e
```

例:

数を集計する

```
>>> $string = "あいうえお";
>>> $n = 0;
>>> foreach (each_byte($string) as $s) { $n += 1; }
>>> $n
=> 15
>>> strlen($string)
=> 15
```

コードポイントごと

コードポイント
（符号位置）

```php
function each_codepoint(string $string)
{
    if (!\preg_match_all('/./su', $string, $matches)) {
        return;
    }

    foreach ($matches[0] as $char) {
        yield $char;
    }
}
```

PHP(コア言語)はUnicodeを
知らないが、PCRE関数
preg_match()はUnicodeを
知ってる（/u 修飾子）

```php
<?php

use function Bag2\iter\string\each_codepoint;

$string = "一二三123あいうABC가나다";

foreach (each_codepoint($string) as $s) {
    echo $s, PHP_EOL;
}
// 一
// 二
// 三
// 1
// 2
// 3
// あ
// い
// う
// A
// B
// C
// 가
// 나
// 다
```

書記素クラスタごと

書記素とは？

人間の目に
1文字に見える
単位

あ

漢

ガ

どれも単独の

書記素

ところでUnicodeに
は「ガ」の複数の
表現方法が存在する

ガ (U+30AC)

ガ (U+30AB U+3099)

ガ (U+FF76 U+FF9E)

ガ（U+30AC）

ガ（U+30AB U+3099）

ガ（U+FF76 U+FF9E）

結合したガ

カ＋゛（結合文字）

半角カ＋゛

Unicode正規化

余談: ファイル名に濁点のついた文字をGitで管理するとMacとそのほかのファイルシステムで諸々の厄介な事態が起こるので禁忌。これはAppleのファイルシステムが伝統的にファイル名をUnicode正規化することが問題

(本筋から逸れるので各自ggr)

書記素クラスタ
のターンはまた
まだ終らないぜ

絵文字が開いてしまった パンドラの匣

Case1: 国旗

絵文字には

国旗が充実

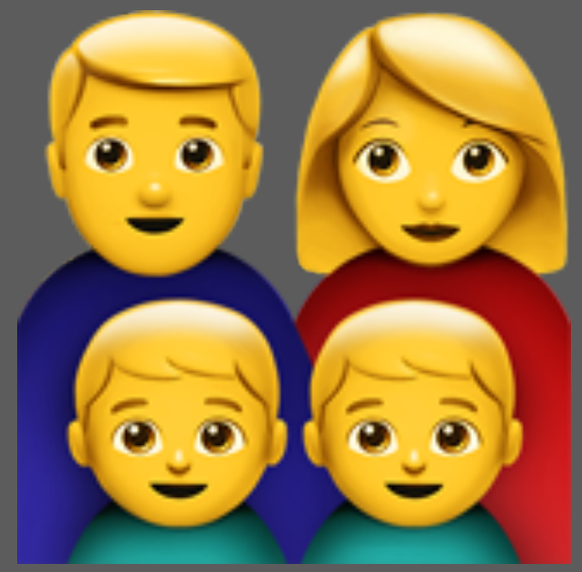
……と
思うじゃん？

実態

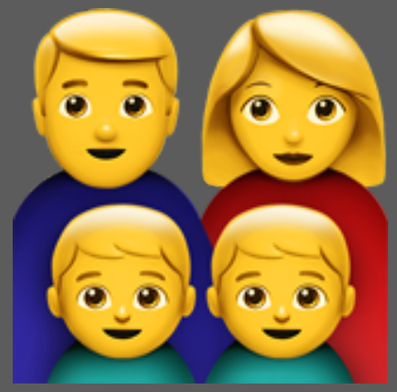
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1F1E-	□	□	□	□	□	□	A	B	C	D	E	F	G	H	I	J
1F1F-	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

国旗専用文字の[J]と[P]を並べて配置すると🇯🇵に化ける

Case2: 家族



この絵文字のバイト数

strlen('  ');

この絵文字のバイト数

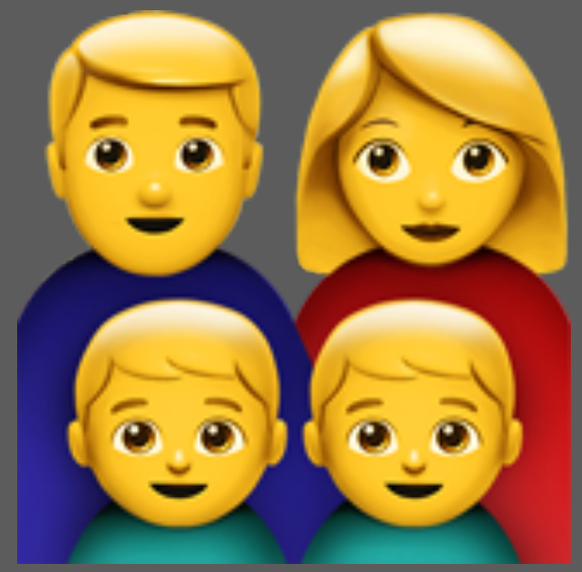
```
strlen('👨‍👩‍👦');
// => 25
```

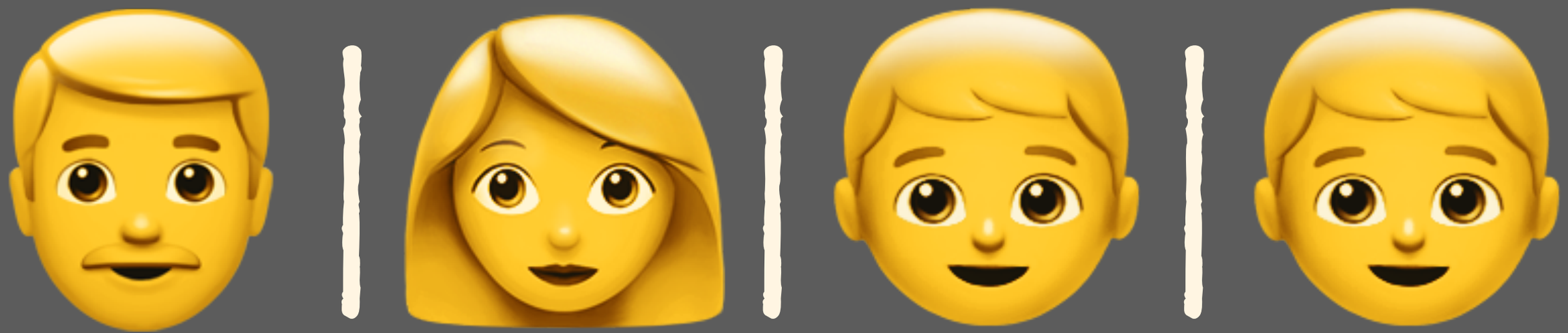
この絵文字のUTF-8文字数

```
mb_strlen('👨‍👩‍👦‍👦', 'UTF-8');
```

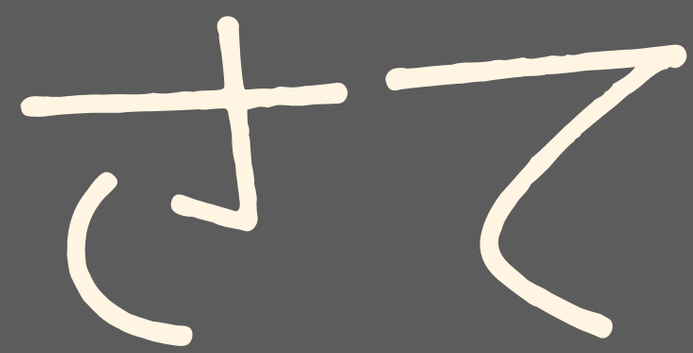
この絵文字のUTF-8文字数

```
mb_strlen('👨‍👩‍👦', 'UTF-8');
// => 7
```





Zero Width Joiner



そんな文字でも

一安心

```php
function each_grapheme(string $string)
{
    $bytes = \strlen($string);

    for ($pos = $next = 0; $next < $bytes; $pos = $next) {
        $len = 1;
        do {
            $chars = \grapheme_extract(
                $string, $len++,
                \GRAPHEME_EXTR_MAXCHARS, $pos, $next
            );
        } while ($pos === $next);

        yield $chars;
    }
}
```

ICU(intl)に依存
(常に正しく計算するにはシ
ステムに最新のUnicode定義
が実装されたICUが必要)

今回の話題は

ただのトリビアか？

No

Webアプリとしては
「文字数」を制限し
なければいけない
場合が多々ある

サービス都合の制限

ストレージ(DB)都合制限

自分がどういう理由で
文字数を制限しようと
しているのか考えよう

例:

MySQLのインデックス長のデフォルト設定が767バイトだからutf8mb4のカラムの文字数をVARCHAR(191)以下にしたい

例:
長いユーザー名は画面
に収まりきらないから
30文字以下にしたい

たいていは

mb_strlen($s, 'UTF-8')

で片がつく

「文字数」と「表示幅」は
全然違う概念なので厄介
（なので、今回紹介した
each_grapheme()を使って計算して
もうまくいかない）

オチはないけど各位
よく考えて「文字」
単位とつきあって
いきましょう

ちなみにTwitterの140文字制限がDBの制約ではないことは明白