Typus Documentation

Release 0.1

Murad Byashimov

Nov 04, 2018

Contents

Typus is a typography tool. It means your can write text the way you use to and let it handle all that formating headache:

```
"I don't feel very much like Pooh today..." said Pooh.
"There there," said Piglet. "I'll bring you tea and honey until you do."
- A.A. Milne, Winnie-the-Pooh

“I don’t feel very much like Pooh today...” said Pooh.
“There there,” said Piglet. “I’ll bring you tea and honey until you do.”
-- A.A. Milne, Winnie-the-Pooh
```

Copy & paste this example to your rich text editor. Result may depend on the font of your choice. For instance, there is a tiny non-breakable space between `A. A.` you can see with Helvetica:

“I don’t feel very much like Pooh today…” said Pooh.
“There there,” said Piglet. “I’ll bring you tea and honey until you do.”
— A. A. Milne, Winnie-the-Pooh

Try out the demo.

CHAPTER 1

Web API

A tiny web-service for whatever legal purpose it may serve.

CHAPTER 2

Installation

```
$ pip install git+git://github.com/byashimov/typus.git#egg=typus
```

CHAPTER 3

Usage

Currently Typus supports English and Russian languages only. Which doesn't mean it can't handle more. I'm quite sure it covers Serbian and Turkmen.

In fact, Typus doesn't make difference between languages. It works with text. If you use Cyrillic then only relative processors will affect that text. In another words, give it a try if your language is not on the list

Here is a short example:

```
>>> from typus import en_typus, ru_typus
...
>>> # Underscore is for nbsp in debug mode
>>> en_typus('"Beautiful is better than ugly." (c) Tim Peters.', debug=True)
'“Beautiful is_better than ugly.” ©_Tim Peters.'
>>> # Cyrillic '' in '()'
>>> ru_typus('" ,  ." ()  .', debug=True)
'« ,  .» ©_ .'
```

The only difference between `en_typus` and `ru_typus` are in quotes they set: “‘’” for English and «„“» for Russian. Both of them handle mixed text and that is pretty awesome.

Typus is highly customizable. Not only quotes can be replaced but almost everything. For instance, if you don't use html tags you can skip `EscapeHtml` processor which makes your Typus a little faster.

CHAPTER 4

What it does

- Replaces regular quotes `"foo 'bar' baz"` with typographic pairs: “`foo ‘bar’ baz`”. Quotes style depends on language and your Typus configuration.
- Replaces regular dash `foo - bar` with mdash or ndash or minus. Depends on case: plain text, digit rage, phone nubers, etc.
- Replaces complex symbols such as `(c)` with unicode characters: ©. Cyrillic analogs are supported too.
- Replaces vulgar fractions `1/2` with unicode characters: ½.
- Turns multiply symbol to a real one: `3x3` becomes `3×3`.
- Replaces quotes with primes: `2' 4"` becomes `2 4`.
- Puts non-breakable spaces.
- Puts ruble symbol.
- Trims spaces at the end of lines.
- and much more.

CHAPTER 5

Documentation

Docs are hosted on readthedocs.org.

See also:

Oh, there is also an outdated Russian article I should not probably suggest, but since all docs are in English, this link might be quite helpful.

CHAPTER 6

Compatibility

Tested on Python 2.6, 2.7, 3.3, 3.4, 3.5, 3.6.

CHAPTER 7

Todo

- Rewrite tests, they are ugly as hell.
- Add missing doctests.

CHAPTER 8

Contents

8.1 What it's for?

Well, when you write text you make sure it's grammatically correct. Typography is *an aesthetic* grammar. Everything you type should be typographied in order to respect the reader. For instance, when you write *"you're"* you put *apostrophe* instead of *single quote*, because of the same reason you place dot at the end of sentence instead of comma, even though they look similar.

Unfortunately all typographic characters are well hidden in your keyboard layout which makes them almost impossible to use. Fortunately Typus can do that for you.

8.1.1 The anatomy

Typus uses *Processors* to do the job and *Mixins* as those settings. And there is a `typus.core.TypusCore` class which makes all of them work together. Here is a quick example:

```
from typus.core import TypusCore
from typus.mixins import EnQuotes
from typus.processors import Quotes

class MyTypus(EnQuotes, TypusCore):
    processors = (Quotes, )

my_typus = MyTypus()
assert my_typus('"quoted text"') == '“quoted text”'
```

`typus.core.TypusCore` runs *typus.processors.Quotes* processor which uses *quotes* configuration from *typus.mixins.EnQuotes*.

8.2 Processors

Processors are the core of Typus. Multiple processors are nested and chained in one single function to do things which may depend on the result returned by inner processors. Say, we set `EscapeHtml` and `MyTrimProcessor`, this is how it works:

```
extract html tags
    pass text further if condition is true
        do something and return
    return the text
put tags back and return
```

In python:

```
from typus.core import TypusCore
from typus.processors import BaseProcessor, EscapeHtml

class MyTrimProcessor(BaseProcessor):
    def __call__(self, func):
        def inner(text, *args, **kwargs):
            # When processor is initiated it gets typus instance
            # as the first argument so you can access to it's configuration
            # any time
            if self.typus.trim:
                trimmed = text.strip()
            else:
                trimmed = text
            return func(trimmed, *args, **kwargs)
        return inner

class MyTypus(TypusCore):
    # This becomes a single function. EscapeHtml goes first
    processors = (EscapeHtml, MyTrimProcessor)

    # Set it `False` to disable trimming
    trim = True

my_typus = MyTypus()
assert my_typus('    test    ') == 'test'
```

Processors can be configured with *Mixins*.

8.2.1 Built-in processors

class typus.processors.**EscapePhrases**(*typus*)

Escapes phrases which should never be processed.

```
>>> en_typus('Typus turns `(c)` into "(c)"', escape_phrases=['`(c)`'])
'Typus turns `(c)` into “©”'
```

Also there is a little helper *typus.utils.splinter()* which should help you to split string into the phrases.

class typus.processors.**EscapeHtml**(*typus*)

Extracts html tags and puts them back after.

```
>>> en_typus('Typus turns <code>(c)</code> into "(c)"')
'Typus turns <code>(c)</code> into “©”'
```

Caution: Doesn't support nested `<code>` tags.

class typus.processors.**Quotes**(**args*, ***kwargs*)

Replaces regular quotes with typographic ones. Supports any level nesting, but doesn't work well with minutes `1'` and inches `1"` within the quotes, that kind of cases are ignored. Use it with *typus.mixins.RuQuotes* or *typus.mixins.EnQuotes* or provide Typus attributes `loq, roq, leq, req` with custom quotes.

```
>>> en_typus('Say "what" again!')
'Say “what” again!'
```

class typus.processors.**Expressions**(**args*, ***kwargs*)

Provides regular expressions support. Looks for `expressions` list attribute in Typus with expressions name, compiles and runs them on every Typus call.

```
>>> from typus.core import TypusCore
>>> from typus.processors import Expressions
...
>>> class MyExpressionsMixin:
...     def expr_bold_price(self):
...         expr = (
...             (r'(\$\d+)', r'<b>\1</b>'),
...         )
...         return expr
...
>>> class MyTypus(MyExpressionsMixin, TypusCore):
...     expressions = ('bold_price', )  # no prefix `expr_`!
...     processors = (Expressions, )
...
>>> my_typus = MyTypus()  # `expr_bold_price` is compiled and stored
>>> my_typus('Get now just for $1000!')
'Get now just for <b>$1000</b>!'
```

Note: *Expression* is a pair of regex and replace strings. Regex strings are compiled with *typus.utils.re_compile()* with a bunch of flags: unicode, case-insensitive, etc. If that doesn't suit for you pass your own flags as a third member of the tuple: `(regex, replace, re.I)`.

8.3 Mixins

Mixins are configurations for *Processors*.

class typus.mixins.**EnQuotes**

Provides English quotes configutation for *typus.processors.Quotes* processor.

```
>>> en_typus('He said "\'Winnie-the-Pooh\' is my favorite book!".')
'He said “‘Winnie-the-Pooh’ is my favorite book!”.'
```

class typus.mixins.**RuQuotes**

Provides Russian quotes configutation for *typus.processors.Quotes* processor.

```
>>> ru_typus(' : "\'-\' --   !".')
' : «„-“ --   !».'
```

class typus.mixins.**EnRuExpressions**

This class holds most of Typus functionality for English and Russian languages. It works with *typus.processors.Expressions*.

expr_abbrs()

Adds narrow non-breakable space and replaces whitespaces between shorten words.

expr_apostrophe()

Replaces single quote with apostrophe.

```
>>> en_typus("She'd, I'm, it's, don't, you're, he'll, 90's")
'She’d, I’m, it’s, don’t, you’re, he’ll, 90’s'
```

Note: By the way it works with any omitted word. But then again, why not?

expr_complex_symbols()

Replaces complex symbols with Unicode characters. Doesn't care about case-sensitivity and handles Cyrillic-Latin twins like с and .

```
>>> en_typus('(c)(C)(r)(R)...')
'©©®®...'
```

Table 1: Character map

...	←	→	±					®	©	Ⓟ	™	SM
...	<-	->	+- or +	<=	>=	/=	==	18.	3.	16.	(tm)	(sm)

expr_del_positional_spaces()

Removes spaces before and after certain symbols.

expr_digit_spaces()

Replaces whitespace with non-breakable space after 4 (and less) length digits if word or digit without comma or math operators found afterwards: 3 apples 40 000 bucks 400 + 3 Skips: 4000 bucks 40 000,00 bucks

expr_linebreaks()

Converts line breaks to unix-style and removes extra breaks if found more than two in a row.

```
>>> en_typus('foo\r\nbar\n\n\nbaz')
'foo\nbar\n\nbaz'
```

expr_math()

Puts minus and multiplication symbols between pair and before single digits.

```
>>> en_typus('3 - 3 = 0')
'3  3 = 0'
>>> en_typus('-3 degrees')
'3 degrees'
>>> en_typus('3 x 3 = 9')
'3 × 3 = 9'
```

(continues on next page)

(continued from previous page)

```
>>> en_typus('x3 better!')
'×3 better!'
```

Important: Should run after *mdash* and *phones* expressions.

---

expr_mdash()
Replaces dash with mdash.

```
>>> en_typus('foo -- bar')  # adds non-breakable space after `foo`
'foo -- bar'
```

expr_pairs()
Replaces whitespace with non-breakable space after 1-2 length words.

expr_phones()
Replaces dash with ndash in phone numbers which should be a trio of 2-4 length digits.

```
>>> en_typus('111-00-00'), en_typus('00-111-00'), en_typus('00-00-111')
('111-00-00', '00-111-00', '00-00-111')
```

expr_primes()
Replaces quotes with prime after digits.

```
>>> en_typus('3\' 5" long')
'3 5 long'
```

Caution: Won't break "4", but fails with " 4".

expr_ranges()
Replaces dash with mdash in ranges. Supports float and negative values. Tries to not mess with minus: skips if any math operator or word was found after dash: 3-2=1, 24-pin. **NOTE**: _range_ should not have spaces between dash: *2-3* and left side should be less than right side.

expr_rep_positional_spaces()
Replaces whitespaces after and before certain symbols with non-breakable space.

expr_ruble()
Replaces and (with or without dot) after digits with ruble symbol.

```
>>> en_typus('1000 .')
'1000 '
```

Caution: Drops the dot at the end of sentence if match found in there.

expr_spaces()
Trims spaces at the beginning and end of the line and remove extra spaces within.

```
>>> en_typus('   foo bar  ')
'foo bar'
```

> **Caution:** Doesn't work correctly with nbsp (replaces with whitespace).

expr_units()
Puts non-breakable space between digits and units.

```
>>> en_typus('1mm', debug=True), en_typus('1mm')
('1_mm', '1 mm')
```

expr_vulgar_fractions()
Replaces vulgar fractions with appropriate unicode characters.

```
>>> en_typus('1/2')
'½'
```

8.4 Utils

typus.utils.**re_compile**(*pattern*, *flags=58*)
A shortcut to compile regex with predefined flags: `re.I`, `re.U`, `re.M`, `re.S`.

Parameters

- **pattern** (*str*) – A string to compile pattern from.
- **flags** (*int*) – Python `re` module flags.

```
>>> foo = re_compile('[a-z]')  # matches with 'test' and 'TEST'
>>> bool(foo.match('TEST'))
True
>>> bar = re_compile('[a-z]', flags=0)  # doesn't match with 'TEST'
>>> bool(bar.match('TEST'))
False
```

class typus.utils.**idict**(*obj=None*, ***kwargs*)
Case-insensitive dictionary.

Parameters

- **obj** (*mapping/iterable*) – An object to initialize new dictionary from
- ****kwargs** – `key=value` pairs to put in the new dictionary

```
>>> foo = idict({'A': 0, 'b': 1}, bar=2)
>>> foo['a'], foo['B'], foo['bAr']
(0, 1, 2)
```

> **Caution:** *idict* is not a full-featured case-insensitive dictionary. As it's made for *map_choices()* and has limited functionality.

typus.utils.**map_choices**(*data*, *group=u'({0})'*, *dict_class=<class 'typus.utils.idict'>*)
typus.processors.Expressions helper. Builds regex pattern from the dictionary keys and maps them to values via replace function.

Parameters

- **data** (*mapping/iterable*) – A pairs of (find, replace with) strings
- **group** (*str*) – A string to format in choices.
- **dict_class** (*class*) – A dictionary class to convert source data. By default *idict* is used which is case-insensitive. In instance, to map `(c)` and `(C)` to different values pass regular python `dict`. Or if the order matters use `collections.OrderedDict`

Returns A regex non-compiled pattern and replace function

Return type tuple

```
>>> import re
>>> pattern, replace = map_choices({'a': 0, 'b': 1})
>>> re.sub(pattern, replace, 'abc')
'01c'
```

typus.utils.**splinter**(*delimiter*)

typus.processors.EscapePhrases helper. Almost like `str.split()` but handles delimiter escaping and strips spaces.

Parameters **delimiter** (*str*) – String delimiter

Raises **ValueError** – If delimiter is a slash or an empty space

Returns A list of stripped phrases splitted by the delimiter

Return type list

```
>>> split = splinter(',  ')  # strips this spaces
>>> split('a, b,c ,  d\,e')  # and this ones too
['a', 'b', 'c', 'd,e']
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index