

Python, Locales and Writing Systems

Rae Knowler
Swiss Python Summit
17th February 2017



About me

CKAN, Symfony, Django

@RaeKnowler

they/their/them



Python 3 is great

Unicode by default!

Source file encoding assumed to be UTF-8

No need to specify `u'foobar'` for non-ascii strings

Less of this:

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
UnicodeEncodeError: 'ascii' codec can't encode  
character u'\xfc' in position 1: ordinal not  
in range(128)
```

Turkish i and ı



Ramazan Çalçoban



Emine Çalçoban

<http://gizmodo.com/382026/a-cellphones-missing-dot-kills-two-people-puts-three-more-in-jail>

Turkish i and ı

Dotless: 'ı' (U+0131), 'İ' (U+0049)

Dotted: 'i' (U+0069), 'İ' (U+0130)

More details here:

<http://www.i18nguy.com/unicode/turkish-i18n.html>

Turkish i and ı

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'tr_TR.utf8')
'tr_TR.utf8'
>>>
>>> turkish_letters = ['ı', 'İ', 'i', 'İ']
```

Turkish i and ı

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'tr_TR.utf8')
'tr_TR.utf8'
>>>
>>> turkish_letters = ['ı', 'I', 'i', 'İ']
>>> print([tl.upper() for tl in turkish_letters])
['I', 'I', 'I', 'İ']
>>> print([tl.lower() for tl in turkish_letters])
['ı', 'i', 'i', 'i']
```

Turkish i and ı - Solutions

- PyICU: a Python extension wrapping IBM's International Components for Unicode C++ library (ICU).

<https://pypi.python.org/pypi/PyICU>

- Or... make a translation table and use `str.translate()` to replace characters when changing the case

Right-to-left writing systems



https://en.wikipedia.org/wiki/Hebrew_language#/media/File:Simtat_Aluf_Batslut.JPG

Right-to-left writing systems

Unicode wants characters ordered **logically**, not **visually**

→ we need bidirectional (bidi) support

→ `pip install python-bidi`

Right-to-left writing systems

```
>>> from bidi.algorithm import get_display
>>>
>>> hebrew_string = 'האקדמיה ללשון העברית'
>>>
>>> get_display(hebrew_string)
'תירבעה ןושלל הימדקאה'
```

Right-to-left writing systems

Arabic letters have
contextual forms:

Their placement in
the text changes
their shape.

General Unicode	Contextual forms				Name
	Isolated	End	Middle	Beginning	
0623 أ	FE83 أ	FE84 آ			'alif
0628 ب	FE8F ب	FE90 ب	FE92 ب	FE91 ب	bā'
062A ت	FE95 ت	FE96 ت	FE98 ت	FE97 ت	tā'
062B ث	FE99 ث	FE9A ث	FE9C ث	FE9B ث	ṭā'
062C ج	FE9D ج	FE9E ج	FEA0 ج	FE9F ج	ġīm

https://en.wikipedia.org/wiki/Arabic_script_in_Unicode#Contextual_forms

Right-to-left writing systems

→ Python Arabic Reshaper to the rescue:

<https://github.com/mpcabd/python-arabic-reshaper>

اللغة العربية



اللغة العربية

Fullwidth and halfwidth characters

Notice any difference?

The quick brown fox jumped over the
lazy dog.

The quick brown fox jumped over the lazy dog.

Fullwidth and halfwidth characters

Courier New doesn't even bother with the fullwidth characters.

T h e q u i c k b r o w n f o x
j u m p e d o v e r t h e l a z y
d o g .

The quick brown fox jumped over the
lazy dog.

Fullwidth and halfwidth characters

Han characters (used in Chinese, Japanese, Korean) are fullwidth:

假借字, 形声字

There are fullwidth and halfwidth kana (Japanese):

ミムメモヤユヨラリルレロワン

ミムメモヤユヨラリルレロワン

Hiragana (Japanese) are always fullwidth:

なにぬねのは

Fullwidth and halfwidth characters

ピリオドは, `s t o p`や`f u l l s t o p`ともいい, 平叙文・命令文の終わりに付ける. 見出しの章・節名や, 図版の説明 (`c a p t i o n`) などでは省いてもよい. しかし, 図版の`c a p t i o n`が2行や3行になるときには付けるようにする.

Fullwidth and halfwidth characters

```
pip install jaconv
```

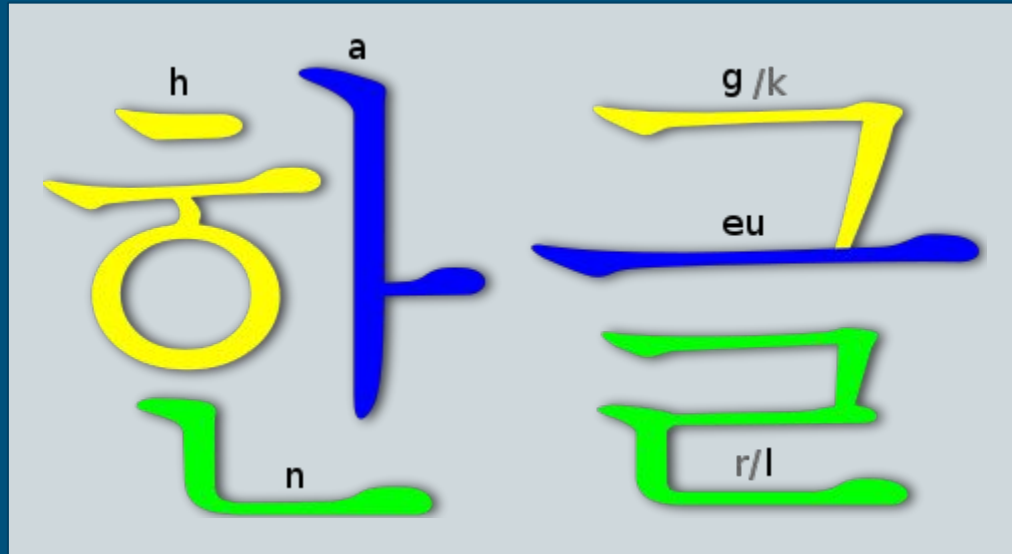
```
>>> import jaconv
>>>
>>> jaconv.z2h('テ ィ ロ ・ フ ィ ナ ー レ')
'テ ィ ロ ・ フ ィ ナ ー レ'
>>>
>>> jaconv.h2z('テ ィ ロ ・ フ ィ ナ ー レ')
'テ ィ ロ ・ フ ィ ナ ー レ'
>>>
```

Fullwidth and halfwidth characters

```
pip install jaconv
```

```
>>> jaconv.h2z('Roman characters', ascii=True)
'R o m a n \u3000 c h a r a c t e r s '
>>>
>>> jaconv.z2h('R o m a n \u3000 c h a r a c t e r s ', ascii=True)
'Roman characters'
>>>
```

Korean text



Lots more detail here:

http://www.gernot-katzers-spice-pages.com/var/korean_hangul_unicode.html

<https://en.wikipedia.org/wiki/Hangul#/media/File:Hangeul.svg>

Korean text

Unicode canonical equivalence:

You can build the same character in several different ways,
and they mean the same thing.

한 means the same as 헀 ㅏ ㄴ

Korean text

Unicode canonical equivalence:

You can build the same character in several different ways,
and they mean the same thing.

한 means the same as ㅎ ㅏ ㄴ

Normal Form D (NFD): ㅎ ㅏ ㄴ

Normal Form C (NFC): 한

Korean text

Unicode compatibility equivalence:

There are multiple code points for identical characters,
for backwards compatibility reasons

U+2160 (ROMAN NUMERAL ONE) is really the same thing as
U+0049 (LATIN CAPITAL LETTER I)

(<https://docs.python.org/2/library/unicodedata.html>)

Korean text

```
>>> korean_string = '한글'
>>>
>>> for ch in korean_string:
...     print(unicodedata.normalize('NFD', ch))
...
한 | ㄱ
ㄱ — ㄴ
```


Korean text

```
>>> for ch in korean_string:  
...     print(unicodedata.normalize('NFC', ch))
```

이
하
의
문
자

Korean text

```
>>> print(unicodedata.normalize('NFD', korean_string))
```

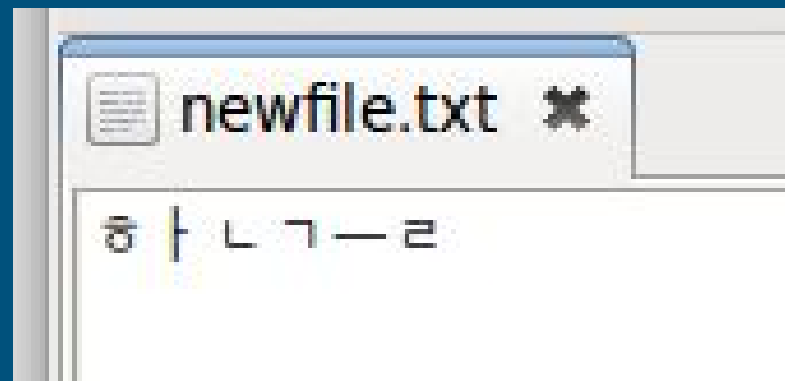
Korean text

```
>>> print(unicodedata.normalize('NFD', korean_string))
```



Korean text

```
>>> print(unicodedata.normalize('NFD', korean_string))
```



Security

This is a huge topic!

A couple of quick examples...

Security

User input:

I don't like raisins

Sanitised user input:

```
'I don\'t like raisins'
```

Hex encoding of \ is 0x5C

Security

Hex encoding for 稞: 0xb8 0x5c

User input:

0xb8' OR 1=1

Sanitised user input:

' 稞 OR 1=1 '

More details here:

<http://howto.hackallthethings.com/2016/06/using-multi-byte-characters-to-nullify.html>

Security - Address Bar Spoofing

A nice google.com link:

<http://google.com/test/test/test/عربي.امارات>

This actually led to:

<http://عربي.امارات/google.com/test/test/test>

More details here:

<http://www.rafayhackingarticles.net/2016/08/google-chrome-firefox-address-bar.html>

Conclusions

This stuff isn't easy ... but it *is* interesting!

There are a lot of useful libraries out there. You won't be the first person to have your particular problem.

Python 3 makes dealing with Unicode a lot easier.

Further links

- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!):
<http://www.joelonsoftware.com/articles/Unicode.html>
- Dark corners of Unicode:
<https://eev.ee/blog/2015/09/12/dark-corners-of-unicode>
- I Can Text You A Pile of Poo, But I Can't Write My Name:
<https://modelviewculture.com/pieces/i-can-text-you-a-pile-of-poo-but-i-cant-write-my-name>