

Linguistically correct Python text rendering

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-02/3050.html>

From: David Opstad (opstad_at_batnet.com)

Date: 02/21/04

Date: Sat, 21 Feb 2004 08:59:59 -0800

I have a question about text rendering I'm hoping someone here can answer. Is there a way of doing linguistically correct rendering of Unicode strings in Python? In simple cases like Latin or Japanese I can just print the string and see the correct results. However, I don't know how to get Python to do the right thing for writing systems which require contextual processing.

For example, let's say I create this Unicode string in Arabic:

```
string1 = u"\u0644\u062e\u0645" # lam khah meem
```

If I print `string1`, I get just the characters I specified above, which is not correct rendering behavior for Arabic. In the simple case, I should get an output string where the order is reversed, and the codes have been changed into their contextually correct forms: initial lam (`\ufedf`), medial khah (`\ufeaa8`) and final meem (`\ufee2`). Optionally, I could even ask for a single ligature combining all three of these; Unicode even encodes this at `\ufd85`.

The situation is even more complicated for writing systems like Devanagari (used in Hindi and Marathi). In this case there are rules for ligatures (called "conjuncts") which are linguistically required, but unencoded by Unicode. Technology was developed 15 years ago to deal with this: fonts contain tables of extra information allowing access to the correct conjunct glyphs at rendering time, even if they're unencoded.

Apple has software that deals with correctly rendering text in cases like this (ATSUI); Microsoft does as well. IBM provides the ICU software classes for this kind of rendering, and I believe FreeType has made a start on dealing with AAT and OpenType fonts as well. So my question is this: how do we get this functionality integrated into Python? I'd love to be able to print any Unicode string and have it come out linguistically correctly, even if I have to do a little extra formatting (e.g. something like adding a new string-formatting conversion character `%t` for typographically rich output).

Any ideas?

Dave Opstad