Insper

08 - Internacionalização e Localização

Desenvolvimento Aberto - 2018/2

Igor Montagner

Na parte expositiva da aula tivemos uma introdução aos problemas de Internacionalização (i18n) e Localização (L10N). Neste roteiro iremos praticar o uso destas técnicas em duas situações: uma aplicação linha de comando de exemplo e em um sistema Web feito em Flask.

Em ambos exemplos vamos trabalhar com o módulo *Babel*, que é feito para facilitar a tradução e localização de aplicações feitas em Python. Outras linguagens de programação possuem bibliotecas similares que seguem a mesma sequência de comandos e usam os mesmos tipos de arquivos.

Sistemas POSIX suportam a determinação de do *locale* utilizado por meio da variável de ambiente *LANG*, que pode ser modificada para cada execução de um programa. O formato padrão usado é `<lingua>_<pais>.<codificacao>`. Para português do Brasil usando codificação UTF8 usamos o locale `pt_BR.utf8`. Rodando o seguinte comando as mensagens de ajuda do `ls` devem aparecer em inglês.

```
LANG=en_US.utf8 ls --help
```

Já executando o comando abaixo elas devem aparecer em português.

```
LANG=pt_BR.utf8 ls --help
```

De maneira mais geral, existe uma série de variáveis *LC_** que controlam qual locale é usado para determinado tipo de dados. Por exemplo, a variável *LC_TIME* define o locale usado para as datas mostradas no sistema. *LC_NUMERIC* é usado para números. A variável *LC_ALL* também pode ser usada e define simultaneamente o mesmo locale para todas as formatações.

Parte 1 - linha de comando

Vamos trabalhar com uma aplicação de linha de comando que nada mais faz que imprimir alguns dados simples como data em extenso, um número fracionário grande e uma mensagem pré-definida. O código completo (arquivo *cli.py*) está abaixo.

```
from datetime import date

from babel.dates import format_date
import locale

print(locale.getdefaultlocale())

if __name__ == '__main__':
    today = date.today()
    print(format_date(today), today)

    print(240000000000.32212)

    name = input('Input your name: ')
    print('Hello {}'.format(name))
```

Uma saída possível seria

```
2018-08-28
240000000000.3221
Input your name: Igor
Hello Igor
```

Como já visto em aula, este programa reúne três das principais saídas que precisam ser formatadas: datas, números fracionários e mensagens para o usuário.

Formatando datas

A formatação de datas é governada para variável `LC_TIME`. O módulo `babel.dates` já possui diversas funções que automaticamente a utilizam para fazer a localização de variáveis do tipo `Date` (usando a função `format_date`) ou `DateTime` (usando `format_datetime`).

Exercício: pesquise como usar estas funções e utilize-as no seu programa para localizar a data por extenso (ou seja, 29 de agosto de 2018).

Formatando números

A formatação de datas é governada para variável `LC_NUMERIC`. O módulo `babel.numbers` possui a função `format_number` que formata um número de acordo com esta configuração.

Exercício: pesquise como usar estas funções e utilize-as no seu programa para localizar o número fracionário mostrado..

Traduzindo mensagens

A parte final consiste em criar traduções das duas strings presentes no texto. A linguagem usada é definida pela variável `LANG`, que pode ser definida separadamente para cada processo. Um dos pontos mais importantes é marcar quais strings deverão ser traduzidas para que uma equipe de tradutores não precise mexer no código. O módulo `gettext` do Python já provê suporte a esta funcionalidade, o *Babel* apenas fornece um conjunto de ferramentas que facilita seu uso.

No começo de todo arquivo Python precisamos importar o `gettext` e depois marcar todas as strings a serem traduzidas com uma chamada à função `gettext.gettext(str)` (sim, `gettext` duas vezes...) Para deixar o código menos sujo é comum importar esta função da seguinte maneira:

```
from gettext import gettext as _
```

Assim, posso marcar todas as strings para serem mostradas com `_` e elas serão marcadas para tradução.

```
print(_("Hello!"))
```
