Insper

07 - Internacionalização e Localização

Desenvolvimento Aberto - 2019/2

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 *LANGUAGE*, 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.

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

Já executando o comando abaixo elas devem aparecer em português.

```
LANGUAGE=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. Veremos a seguir como usar `LC_TIME` e `LC_NUMERIC` para controlar como datas e números são exibidos e `LANGUAGE` para definir a lingua de exibição de um programa.

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.

```
import babel
from datetime import date
from datetime import date, datetime, time
from babel.dates import format_date, format_datetime, format_time
from babel.numbers import format_number, format_decimal, format_percent

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

    number = 240000000000.32212
    print(number)
    number = format_number(number)
    print(number)


    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).

Exercício: o quê acontece quando definimos a variável de ambiente `LC_TIME=en_US.utf8` e rodamos o programa? E se usamos `LC_TIME=pt_BR.utf8`?

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.

Exercício: teste seu programa com `LC_NUMERIC=en_US.utf8` e `LC_NUMERIC=pt_BR.utf8`. Os efeitos são os esperados?s

Traduzindo mensagens

A parte final consiste em criar traduções das duas strings presentes no texto. A linguagem usada é definida pela variável `LANGUAGE`, 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.

A implantação do framework de tradução é feita em quatro passos:

1. Marcação das strings a serem traduzidas
2. Extração destas strings do código em um arquivo modelo `.pot`
3. Criação de traduções `.po` a partir do modelo criado no passo anterior
4. Compilação das strings traduzidas em um arquivo binário `.mo`

No arquivo principal de nossa aplicação podemos "instalar" o framework de tradução e marcar todas nossas strings a serem traduzidas com a função `_()`. A instalação é feita pelo seguinte trecho de código.

```
import gettext
gettext.install('cli', localedir='locale')
# cli é o nome do arquivo em que guardamos nossas traduções
# localedir é o caminho onde estão armazenadas as traduções. Pode ser um caminho relativo.
```

Devemos então marcar todas as strings para serem traduzidas com `_()`. Podemos usar `_()` em qualquer arquivo do projeto, mesmo que a instalação tenha sido feita somente no arquivo principal.

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

Os passos seguintes são feitos com auxílio do *Babel*, que efetivamente analisa nosso código Python e extrai as strings para tradução. A criação do arquivo modelo de tradução a partir dos arquivos do diretório atual é feita com o seguinte comando.

```
$ pybabel extract . -o cli-model.pot
```

Criamos então uma nova tradução usando o seguinte comando. A opção `-D` indica o nome do arquivo em que as traduções serão guardadas (usado em `gettext.install`). A opção `-l` indica o locale da tradução. A opção `-d` indica o `localedir` usado em `gettext.install`.

```
$ pybabel init -i cli-model.pot -D cli -l pt_BR -d locale
```

Devemos então editar o arquivo criado em *locale/pt_BR/LC_MESSAGES/cli.po*. Serão apresentados (após algumas linhas de comentários) pares de linhas como as seguintes. O primeiro valor `msgid` é a string a ser traduzida e o segundo `msgstr` é a tradução no locale *pt_BR* (pois o arquivo está na pasta pt_BR do localedir).

```
msgid "Input your name: "
msgstr ""
```

Apesar de ser possível fazer tudo diretamente no arquivo de texto, é mais conveniente usar softwares como o poedit ou este editor online(https://localise.biz/free/poeditor).

Com as strings traduzidas vamos finalmente compilar nossos resultados. Isto é feito para que não seja possível mexer nos arquivos de tradução em uma versão *Release* do programa.

```
$ pybabel compile -D cli -l pt_BR -d locale
```

Tudo pronto!

Podemos definir a variável `LANGUAGE` para modificar a lingua de um programa (como visto anteriormente com `ls`). Execute seu programa diretamente e depois setando `LANGUAGE=pt_BR.utf8`. Os resultados foram os esperados?

Valide sua solução com o professor neste momento. Se tudo estiver correto ele irá adicionar a skill *Tradução básica* em seu usuário.

Parte 2 - Traduzindo no mundo real

Agora que você já conhece os passos necessários para traduzir um software é o momento de colocar esse conhecimento em prática. O trabalho de seu grupo será encontrar softwares que necessitem de traduções para o

Português brasileiro e realizá-las. Alguns projetos também disponibilizam arquivos *.po para tradução de guias de usuário e isto também é válido neste item.

- GNOME - sistema de traduções
- KDE - status de traduções e brasileiros tradutores.
- Inkscape
- Openshot

Ao ter traduções aceitas por um projeto vocês receberão a skill *Tradução aceita!*. Cada membro do grupo deverá mandar ao menos um "pacote" de traduções, seja de strings de UI ou guias de usuário/documentações traduzidas. Contanto que algo seja feito manipulando arquivos *.po está valendo.